Raisonner avec les machines : la programmation est-elle une forme de connaissance ?

Henri Stephanou

UNIVERSITÉ PARIS 1
PANTHÉON SORBONNE

UNIVERSITÉ PARIS 1 PANTHÉON-SORBONNE
École doctorale de philosophie
Institut d'histoire et de philosophie des sciences et des techniques

THÈSE
pour l'obtention du titre de Docteur en philosophie
présentée et soutenue publiquement
le 18 décembre 2023 par

Henri Stéphanou (Salha)

Raisonner avec les machines
La programmation est-elle une forme de connaissance ?

Sous la direction de M. Pierre Wagner, professeur des universités

Membres du jury :

Mme Liesbeth De Mol	CNRS, chargée de recherche	
M. Daniel Jackson	MIT, professeur	rapporteur
M. Baptiste Mélès	CNRS, chargé de recherche	
M. Dominique Pradelle	Sorbonne-Université, professeur	
Mme Sophie Roux	ENS-PSL, professeure	rapporteuse
M. Ray Turner	University of Essex, professeur émérite	
M. Pierre Wagner	Université Paris 1, professeur	directeur

Institut d'histoire et de philosophie des sciences
et des techniques
13, rue du Four
75006 Paris

Université Paris 1 Panthéon-Sorbonne
École doctorale de philosophie
13, rue du Four
75006 Paris

C'est en vertu de la capacité à revenir que lui donne le fil d'Ariane, elle demeurée au seuil, que [Thésée] pourra sortir du labyrinthe. Il pourra en sortir dans la stricte mesure où il n'en aura jamais tout à fait quitté l'entrée.

Pierre-Marie Hasse
Le cercle sur l'abîme, p. 400

Résumé et sommaire analytique

Résumé

Ce travail propose une interprétation nouvelle de l'effectivité qu'a l'informatique à transformer notre monde humain. Cette interprétation se passe des notions habituelles de *calcul* et d'*information* et se fonde plutôt sur l'idée que les investigations, par lesquelles nous résolvons nos problèmes (qu'ils soient théoriques ou pratiques), peuvent être récapitulées en des procédures qui expliquent comment éviter l'effort d'une seconde recherche. Lorsque ces procédures sont exprimées par des règles exactes, c'est-à-dire prédéfinies dans un système, et que celles-ci parcourent exhaustivement l'ensemble des conditions possibles du problème, alors elles sont *systématiques* et peuvent être *programmées*. Une partie importante de notre travail est consacrée à l'établissement d'une telle épistémologie, qui interroge en particulier l'histoire de la philosophie sur les notions de *savoir pratique, investigation, règle, procédure, machine et système*, d'Aristote à Dewey.

L'idée de *procédure systématique* précède l'informatique ; elle est caractéristique de l'approche industrielle du travail, qui le décompose analytiquement en tâches élémentaires et le recompose, soit pour la spécialisation, soit pour l'automatisation. Le fait fondamental ici est la séparation complète de la *conception* du travail et de son *exécution*, où tout besoin de jugement doit être aboli. La géométrie, qui construit ses figures et en prouve la justesse par des instructions exactes, en donne le modèle, et inspire la figure de l'ingénieur moderne. En particulier, la machine est tout phénomène dynamique dont l'interaction (en général *pauvre*) avec son environnement semble pouvoir être décrite systématiquement. Elle est donc le lieu idoine d'exécution des procédures de l'ingénieur.

Il est possible de construire une machine universelle (c'est-à-dire capable d'exécuter des procédures quelconques, du moment qu'elles sont décrites dans un système de règles, quel qu'il soit) dans des *contrôles de transmission* (système d'irri-

gation, mécanisme d'horlogerie, circuit électrique, etc.).

Programmer, c'est insérer une telle machine dans une situation afin d'y résoudre un problème. La machine rend une règle effective, c'est-à-dire immédiatement valide dans la situation. Lorsque celle-ci est symbolique (une investigation en cours) la machine non seulement accélère la recherche, mais également lui donne accès à de nouvelles opérations, permet de vérifier le raisonnement, d'explorer des espaces gigantesques de règles, etc. Lorsque la situation est l'expérience elle-même, la programmation permet de rendre dynamiques nos systèmes de règles institutionnels et techniques (par exemple le fonctionnement d'une administration ou d'une usine). Ces deux formes d'effectivité – au sein de l'investigation elle-même, et au sein des cadres institutionnels et techniques de nos pratiques – expliquent les bouleversements qu'occasionne l'informatique dans notre monde humain.

Structure générale de l'argumentation

Notre argumentation se développe en cinq parties, de deux chapitres chacune, sauf la troisième qui en comporte trois. Il y a donc en tout onze chapitres, auxquels s'ajoutent une introduction et une conclusion.

La première partie dégage une interprétation préliminaire de ce qu'est programmer, sur la base d'observations de cette activité, d'exemples et d'une revue des études sur le sujet.

Les trois parties centrales établissent les thèses présentées dans les deux premiers paragraphes du résumé ci-dessus, et portent donc sur l'épistémologie de l'investigation, des procédures et des systèmes de règles.

La dernière partie établit les thèses présentées dans les deux derniers paragraphes du résumé.

Dans la présentation analytique que nous faisons ci-après de notre argumentation, les indications comme ***3.5*** désignent le chapitre et la section correspondant aux arguments qui les suivent.

1 Introduction

(1.1) Nous commençons par expliciter note étonnement concernant l'effectivité des ordinateurs dans notre monde humain, ce que nous entendons par *effectivité*, et pourquoi cet étonnement n'est pas aisé à dissiper par de simples explications techniques. ***(1.2)*** Nous étudions la thèse selon laquelle les ordinateurs seraient

effectifs parce qu'ils servent à « traiter l'information » et montrons son insuffisance si on n'explique pas d'abord *ce qu'est programmer*. ***(1.3)*** Nous étudions certains rapprochements qui ont déjà été faits entre programmation et connaissance, notamment l'hypothèse computationnelle de l'esprit il y a une cinquantaine d'années et le mouvement de la pensée computationnelle (*computational thinking*) aujourd'hui. ***(1.4)*** Nous parvenons à l'hypothèse que, si programmer est un acte de connaissance, alors elle aurait trait aux méthodes ou aux procédures, mais une telle idée nous semble problématique. ***(1.5)*** Nous établissons des connexions entre ce problème et plusieurs questions actuelles de philosophie et de sciences humaines, concernant le rapport entre connaissance théorique et pratique ainsi qu'entre sciences et techniques, le statut des règles dans la philosophie du langage ordinaire, la structure de la cognition. ***(1.6)*** Nous situons également ce problème dans le champ dynamique de la philosophie de l'informatique.

Partie I

Qu'est-ce que programmer ?

2 Premières approches

Nous introduisons ce chapitre par l'annonce provocante, faite récemment, de la *fin de la programmation*, que l'apprentissage machine serait amené à remplacer. Cette thèse permet de poser dans toute son acuité la question de ce qu'on entend par *programmer* dans ce genre d'assertions. ***(2.1)*** Nous commençons par observer la programmation comme une activité socio-économique, et montrons l'insuffisance *structurelle* des statistiques à cerner cette activité aux frontières fluides. ***(2.2)*** Une expérience concrète et simple de programmation faite par l'auteur est présentée, ainsi que les difficultés qu'il y a rencontrées. Cela permet de dégager la complexité propre à cette activité : celle d'atteindre une forme de *systématicité*. ***(2.3)*** Nous contrastons cet exemple avec un autre, tiré de la programmation professionnelle. Cela nous amène à donner un bref aperçu des méthodes professionnelles de développement logiciel, où nous insistons sur leur évolution rapide et sur leur diversité. ***(2.4)*** Afin de mesurer l'ampleur de la diversité que recouvre la programmation, nous dégageons quatre tendances observables de sa transformation dans des formes nouvelles. ***(2.5)*** Cette mesure de diversité étant prise, nous

concevoir un bâtiment, de chercher la solution d'une équation, etc. – en tant qu'elle consiste à organiser une situation problématique en parties correctement agencées entre elles, de sorte que leur interaction produise l'effet attendu. *(**4.2**)* Or l'ordinateur est une machine où de tels schémas, représentés par des chaînes de symboles, peuvent être reconfigurés à volonté, grâce à la programmation. Et comme un programme, une fois écrit, devient un objet qui peut être archivé, analysé, transmis, réutilisé, la programmation est donc amenée, selon Simon, à devenir la *lingua franca* de toutes les activités de résolution de problème et de toutes les professions (médecine, droit, ingénierie, gestion, administration, etc.), qui peuvent se constituer enfin comme sciences. *(**4.3**)* Nous explicitons les résultats importants de cette thèse pour notre recherche, mais également ses difficultés, qui tiennent selon nous à l'hypothèse computationnelle de l'esprit, dont Simon fut l'un des grands promoteurs, c'est-à-dire à l'idée que toute activité rationnelle procéderait par application de règles. *(**4.4**)* Ces difficultés nous amènent à développer une réflexion sur la différence qu'il y a entre *règle technique* et *sens pratique*. Nous partons d'une étude terminologique des mots *pratique* et *technique* et la prolongeons par la lecture du philosophe éthique Alasdair MacIntyre et de Pierre Bourdieu ; cette enquête nous permet de dégager deux problèmatiques bien distinctes concernant les rapports entre savoir pratique, savoir technique, et science.

5 Raison pratique et logique industrielle

*(**5.1**)* Nous approfondissons la notion de rationalité pratique à l'aide d'études d'anthropologie du travail concernant l'usage des règles et des procédures en milieu technique, ainsi que *(**5.2**)* des exemples de conflits entre les règles rigides des systèmes d'information et le fonctionnement fluide des organisations. *(**5.3**)* Nous avançons que la logique procédurale stricte, loin d'être naturelle aux personnes, est caractéristique des seuls milieux industriels. Cela nous amène à nous interroger sur la proximité conceptuelle entre informatique et industrie, qui est également historique (avec Babbage) ; nous les trouvons toutes deux caractérisées par une même démarche analytique et systématique des problèmes à résoudre. *(**5.4**)* Cela nous permet de dégager une typologie des *avantages relatifs* d'effectivité qu'apporte l'informatique, qui sont similaires à ceux de l'industrie, et qui sont loin de se le limiter aux seuls gains de productivité. *(**5.5**)* Nous avançons ainsi l'hypothèse que les méthodes informatiques et industrielles ne sont pas des savoirs pratiques ou techniques qui seraient caractérisés par une simple attention accrue à la rigueur

ou à la précision de leurs règles, mais des savoirs d'une forme radicalement différente, qu'on a souvent d'ailleurs qualifiés de « scientifiques ». Mais dans ce cas, comment expliquer que ces savoirs se trouvent dotés d'une telle *effectivité*, s'ils sont en *rupture épistémique* avec ceux par lesquels les personnes transforment leur monde ? ***(5.6)*** Nous commençons l'exploration de cette hypothèse difficile en examinant la conception particulièrement originale que Ryle avait du savoir théorique, et qui nous invite à prendre le concept problématique de *systématicité* comme fil directeur de notre questionnement.

Partie III

Théorie et pratique

Nous nous posons dans cette partie la question de la possibilité d'un savoir « systématique » qui porterait sur la résolution de problème, quels qu'ils soient, y compris s'ils sont de nature pratique ou technique. Nous abordons cette question en nous demandant quelles transformations du concept même de *savoir* ont été nécessaires afin que cette possibilité puisse devenir *pensable*.

6 Délibération et système

Nous commençons par approfondir le caractère paradoxal d'une telle idée en analysant la distinction que fait Aristote entre la raison scientifique qui porte sur les principes nécessaires des choses, et la raison délibérative qui s'occupe des objets contingents de la volonté. Cette distinction *générique* semble dénier à la seconde la possibilité de s'exprimer sous la forme d'un système déductif (et donc, sans doute, d'être systématique). Notre analyse se concentre sur la technique, dont nous étudions ***(6.1)*** d'une part, sa proximité et sa différence avec la science, puis ***(6.2)*** avec la prudence. Cela nous amène à proposer une interprétation originale du statut de la technique chez Aristote. ***(6.3)*** La question devient alors de savoir à quelles conditions la technique pourrait être systématique, ce qui nous amène à étudier une méthode d'investigation qu'Aristote présente comme « universelle » au livre I des *Premiers analytiques*, et que la tradition a nommée *pont aux ânes*. ***(6.4)*** Notre conclusion permet d'évaluer de manière nuancée le scepticisme d'Aristote quant à la possibilité de systèmes techniques.

7 Machine et système aux Temps modernes

Nous tentons de poser dans ce chapitre quelques jalons quant à l'émergence, dans l'histoire, de la conception de cette possibilité. *(7.1)* Nous commençons par rappeler les nombreuses transformations que le couple théorie / pratique connaît dès le haut Moyen Âge, et qui en atténuent l'opposition. Nous nous intéressons ensuite au grand mouvement de rationalisation des pratiques que sont, aux Temps modernes, les *réductions en art*, c'est-à-dire ces tentatives de description raisonnée d'un métier ou d'un art, qui passe par l'explicitation de ses règles et de sa rationalité sous-jacente. *(7.2)* Nous étudions les limites des formes systématiques que prennent ces exposés et les contrastons avec celle de la géométrie, *(7.3)* qui est au cœur de la pratique de l'ingénieur militaire, lui permettant d'envisager une conception *totale* de son ouvrage, qui supprime toute initiative et décision dans l'exécution. *(7.4)* Nous nous intéressons particulièrement au statut singulier des machines, à la fois objets de conception et modèles explicatifs des phénomènes naturels. *(7.5)* Nous mettons ce statut en perspective des rapports étroits qu'entretient la mécanique avec la géométrie aux Temps modernes. *(7.6)* Nous achevons ce chapitre par l'examen de l'émergence de la notion de *système* et de sa critique au XVIII^e^ siècle.

8 Les règles du jugement et de l'action

Ce chapitre est consacré à mettre en dialogue Kant et Aristote sur le statut du savoir technique, que le philosophe allemand présente, dans sa célèbre introduction à la *Critique de la faculté de juger*, comme un « corollaire » de la connaissance théorique. *(8.1)* Nous analysons le sens d'une telle assertion en faisant appel au rôle central que joue la notion de *règle* dans la *Critique de la raison pure*. *(8.2)* Nous présentons des objections d'inspiration aristotélicienne à cette réduction de la rationalité pratique à celle de l'entendement théorique, d'abord celle d'Anscombe, puis *(8.3)* un argument modal qui oppose la description de la situation à la prescription de l'option, enfin celle de philosophes récents de la technique, qui insiste sur les formes et finalités particulières du savoir technique. *(8.4)* Nous avançons que ces arguments, malgré leur force, échouent à remettre en cause la position kantienne. Cependant, celle-ci présente une nouvelle difficulté pour notre recherche, du fait que Kant confine la notion de *système* à sa fonction architectonique dans l'unification des connaissances à laquelle aspire la raison. On ne

voit pas alors, en effet, quel rôle positif elle pourrait jouer dans les investigations mêmes de l'entendement, et donc ce que pourrait signifier la résolution *systématique* de problème qui anime (selon notre hypothèse) les méthodes de l'industrie et de l'informatique.

Partie IV

La résolution de problème

Dans cette partie, sur la base des acquis de nos deux parties précédentes, nous cherchons à dégager ce que pourrait être une *logique de l'investigation* qui permettrait d'expliquer sa possibilité de prendre une forme systématique.

9 Construire et prouver

Nous commençons par prendre le fil conducteur des mathématiques, qui nous sont apparues (en particulier la géométrie) comme le lieu par excellence de la résolution systématique de problème. ***(9.1)*** Nous nous intéressons d'abord à la notion de problème elle-même et à la dualité classique qu'on présente entre problèmes de *prouver* une proposition et problèmes de *trouver* un objet (un nombre, une fonction, une figure, etc.). Nous présentons la solution offerte par la théorie constructive des types, de Per Martin-Löf, pour résorber une telle distinction. Elle permet selon nous de repenser la notion de problème à neuf, selon des distinctions complètement différentes. ***(9.2)*** Nous testons l'applicabilité d'une telle théorie à la géométrie euclidienne, et notamment l'interprétation qu'elle permettrait de dégager du débat antique, rapporté par Proclus, concernant la primauté qu'il faudrait accorder, dans les *Éléments*, aux *théorèmes* ou aux *problèmes*. ***(9.3)*** Après l'étude des tentatives déjà réalisées sur cette question, nous esquissons une solution originale qui permet d'illustrer la complexité potentielle des structures de problèmes. ***(9.4)*** Nous sommes alors en mesure de proposer une caractérisation de la résolution systématique de problème selon trois principes. ***(9.5)*** Notre recherche n'est cependant pas achevée car les mathématiques n'expliquent pas le rapport qui existe entre l'investigation elle-même et les solutions systématiques de problème auxquelles elle parvient. Dans les termes de notre sujet, c'est comme si elles nous donnaient les *programmes*, mais ne nous disaient rien de la *programmation*.

10 La théorie de l'investigation

Nous adressons cette question à la *théorie de l'investigation* de John Dewey, qui en a fait le cœur de sa philosophie de la connaissance. ***(10.1)*** Nous rappelons brièvement les concepts centraux de cette théorie, notamment ceux de signification et de langage, ***(10.2)*** puis ceux de jugement et d'expérience, avant de présenter sa conception de la résolution de problème. Cela nous permet d'exposer son interprétation ***(10.3)*** de la nature des mathématiques, ***(10.4)*** des théories scientifiques, ainsi que ***(10.5)*** des systèmes institutionnels et techniques. Nous avançons que la théorie constructive des types permet de compléter de manière convaincante cette épistémologie, et qu'ensemble elles permettent de penser à la fois la continuité et la rupture qui existe entre les investigations pratiques et techniques d'une part, et les investigations systématiques. ***(10.6)*** Nous présentons une synthèse générale des acquis épistémologiques des trois dernières parties.

Partie V

Machines et systèmes

Les trois parties précédentes ont permis de dégager un cadre épistémologique minimal autour de la notion de *résolution systématique de problème*. Nous achevons d'élucider, dans cette dernière partie, la nature de la programmation, notamment par le biais d'une réflexion sur la notion de *machine*, ainsi que les raisons de son effectivité dans notre monde humain.

11 La programmation de machines

(11.1) Nous commençons par analyser quel rapport existe entre programmation et investigation systématique, en prenant pour fil directeur l'analyse de de la notion de *procédure effective* en théorie de la calculabilité. ***(11.2)*** Nous montrons que cette relation fondamentale permet de poser les bases d'une épistémologie de l'informatique, dans ses divisions principales (algorithmique et méthodes formelles) ainsi que dans ses dimensions théorique et pratique. ***(11.3)*** Nous nous interrogeons à présent sur le lien nécessaire qui doit exister entre ce concept de la programmation et celui de *machine*, en questionnant l'exigence répétée des mathématiciens que le calculateur se comporte *mécaniquement*. Cela nous amène à

proposer une définition nouvelle de la notion de machine, comme *phénomène dynamique dont l'interaction avec son environnement semble pouvoir être décrite par des règles*. ***(11.4)*** Nous montrons la possibilité logique de construire un ordinateur grâce à ce que nous appelons un mécanisme de *contrôle de transmissions*, ***(11.4)*** et nous expliquons en quel sens certaines de ces machines peuvent être dite *universelles*. ***(11.5)*** Nous approfondissons la notion de mécanisme grâce à la confrontation des thèses du pan-computationnalisme, une doctrine métaphysique, et du néo-mécanisme, une doctrine épistémologique. ***(11.6)*** Nous dégageons enfin les manières par lesquelles la programmation transforme les investigations systématiques des mathématiques, des sciences et de l'ingénierie.

12 Rendre les règles effectives

Nous explorons enfin l'effectivité des ordinateurs hors du champ de l'investigation, au sein même des pratiques humaines. ***(12.1)*** Nous clarifions le concept d'*interactivité*, qui est central en informatique théorique ***(12.2)*** et montrons en quelle mesure une procédure interactive est capable de créer un *couplage* entre machine et situation, qui rend les règles programmées *directement effectives*. ***(12.3)*** Cette possibilité infiltre le fonctionnement de nos systèmes objectifs de règles, institutionnels et techniques, et les rend ainsi *dynamiques*. ***(12.4)*** Nous montrons en quoi ce nouveau type de situations, quoique *systématique*, est porteur du risque d'un nouveau désordre.

13 Conclusion

(13.1) Nous reprenons l'ensemble des questions posées par ce travail et leur apportons une réponse succincte. Deux questions importantes restent cependant ouvertes, celle du statut de l'apprentissage machine statistique et celle de la nature de l'effectivité des machines. ***(13.2)*** Nous mobilisons notre cadre épistémologique pour analyser l'apprentissage machine statistique et suggérer qu'il permet d'étendre le domaine de la systématicité à des formes de problèmes jusqu'ici inaccessibles à la programmation. ***(13.3)*** Nous précisons le sens en lequel il faut comprendre l'*effectivité des machines*, et comment elle dérive entièrement de l'*effectivité de la connaissance* elle-même. ***(13.4)*** Nous montrons enfin en quelle mesure notre travail peut éclairer les débats et éthiques et politiques liés aux bouleversements que le numérique engendre.

Mots clés

Aristote – Kant – Dewey – Ryle – Jackson (Michael) – Simon (Herbert) – Martin-Löf– Machine – Règle – Procédure – Système – Investigation – Pratique – Technique – Programme – Informatique

Abstract and Analytical Summary

Abstract

This study proposes a new interpretation of the effectiveness of computing in transforming our human world. This interpretation dispenses with the usual notions of *computation* and *information* and is based instead on the idea that the inquiries, by which we solve our problems (whether theoretical or practical), can be summed up in procedures that explain how to avoid the effort of a second search. When these procedures are expressed by exact rules, *i.e.* predefined in a system, and when these rules run exhaustively through the set of possible conditions of the problem, then the procedures are *systematic* and can be *programmed*. A major part of our work is devoted to establishing such an epistemology, which in particular inquires into the history of philosophy on the notions of *practical knowledge, inquiry, rule, procedure, machine and system*, from Aristotle to Dewey.

The idea of a *systematic procedure* predates computer science; it is characteristic of the industrial approach to work, which breaks it down analytically into elementary tasks and reconfigures it, either for specialization or for automation. The fundamental fact here is the complete separation of the *design* of work from its *execution*, in which all needs for judgment must be eliminated. Geometry, which constructs its figures and proves their correctness through exact instructions, provides the model for such a design, and inspires the figure of the modern engineer. In particular, the machine is any dynamic phenomenon whose interaction (generally *poor*) with its environment can be described systematically. It is therefore the ideal place for carrying out the engineer's procedures.

It is possible to build a universal machine (*i.e.* capable of executing any procedures, as long as they are described in a system of rules, whatever that may be) in *transmission controls* (irrigation system, clock mechanism, electrical circuit, etc.).

To program means to insert such a machine into a situation in order to solve a

problem. The machine makes a rule effective, *i.e.* immediately valid in the situation. When the situation is symbolic (an inquiry in progress), the machine does not only speed up the search, but also gives access to new operations, allows to verify the reasoning, to explore gigantic rule spaces, and so on. When the situation is experience itself, programming makes our institutional and technical rule systems dynamic (for example, the operation of an administration or a factory). These two forms of effectiveness - within the inquiry itself, and within the institutional and technical frameworks of our practices - explain the upheavals that computing is causing in our human world.

General structure of the argument

Our argument develops in five parts, each one having two chapters, except the third, which has three. There are therefore eleven chapters in all, plus an introduction and a conclusion.

The first part gives a preliminary interpretation of what programming is, based on observations of this activity, examples and a review of studies on the subject.

The three central parts establish the theses presented in the first two paragraphs of the abstract. They deal with the epistemology of inquiry, procedures and systems of rules.

The final part establishes the theses presented in the last two paragraphs of the abstract.

In the following analytical presentation of our argument, indications such as ***3.5*** refer to the chapter and section corresponding to the arguments that follow them.

1 Introduction

(1.1) We analyse first our surprise about the effectiveness of computers in our human world. We explicit what we mean by *effectiveness*, and why this puzzlement cannot be easily dispelled by simple technical explanations. ***(1.2)*** We study the thesis that computers are effective because they are used to "process information" and show its inadequacy if we do not first explain *what programming is*. ***(1.3)*** We analyse some theses connecting programming and knowledge, notably the *computational hypothesis of the mind* some fifty years ago and the *computational thinking* movement today. ***(1.4)*** We arrive at the hypothesis that, if

programming is an act of knowledge, then its objects would be methods or procedures. However, such an idea seems problematic to us. *(1.5)* We establish connections between this problem and several contemporary questions in philosophy, notably concerning the relationship between theoretical and practical knowledge as well as between science and technology, the status of rules in the philosophy of ordinary language, the structure of cognition. *(1.6)* We also situate this problem in the dynamic field of the philosophy of computer science.

Part I

What is programming?

2 First approaches

We introduce this chapter with the provocative announcement, made recently, about the looming *end of programming*, which could be replaced by machine learning. This thesis allows us to pose the problem of *what is exactly meant* by *programming* in such phrases. *(2.1)* We observe programming as a socio-economic activity, and show the *structural* inadequacy of statistics to capture this activity with fluid boundaries. *(2.2)* A concrete and simple programming experiment made by the author is presented, as well as the difficulties he encountered there. This allows to bring out the complexity inherent in this activity: that of achieving a form of *systematicity*. *(2.3)* We contrast this example with another, taken from professional programming. We give a brief overview of professional software development methods. We emphasise their rapid evolution and diversity. *(2.4)* In order to measure the extent of the diversity that programming encompasses, we identify four observable trends in its transformation into new forms. *(2.5)* Having taken this measure of diversity, we present a definition of programming that covers all the realities observed, and *(2.6)* submit it to critical discussion.

3 To reason with machines

This chapter aims to make explicit the process of *knowledge* that takes place during programming. We first outline *(3.1)* the classical interpretation of this process, which sees it composed of three moments (specification, coding, implementation). We outline the historical background of this interpretation and the heated

discussions it triggered. *(3.2)* We consider it unsatisfactory for specific reasons, which we make explicit through the study of several examples of programming that it struggles to account for. *(3.3)* We propose an alternative interpretation, which is based on a recognised programming method developed by the computer scientist Michael Jackson. It leads us to make explicit our first thesis, that *to program is to insert a machine into a situation in order to solve a problem*. *(3.4)* We show the originality of this thesis and submit it to critical discussion.

Part II

The general form of practical reason?

The assertion of this thesis requires us, on the one hand, to understand what it is *to solve a problem* (the *genus* of knowledge that programming is), and on the other hand what is a machine (its *specific difference*). These questions guide our epistemological reflection and occupy the three central parts of this work.

4 Design and deliberation

The method we have just assigned ourselves: understanding programming from problem solving, could be challenged. We need to take the opposite approach, Herbert Simon argues in substance, because programming gives the *exact* form of problem solving. Therefore problem solving is best understood from the perspective of programming itself. *(4.1)* We begin by making explicit the context and meaning of such an assertion, which forms the core of Simon's celebrated work *The Sciences of the Artificial*. He calls *design* any deliberation – whether prescribing a remedy, planning a troop movement, designing a building, seeking the solution to an equation, etc. – insofar as it consists in organising a problematic situation into correctly arranged parts, so that their interaction produces the expected effect. *(4.2)* Now the computer is a machine where such schemes, represented by strings of symbols, can be reconfigured at will, thanks to programming. And since a programme, once written, becomes an object that can be archived, analysed, transmitted, reused, programming is therefore bound, according to Simon, to become the *lingua franca* of all problem-solving activities and all professions (medicine, law, engineering, management, administration, etc.), which can

finally constitute themselves as sciences. *(4.3)* We make explicit the important results of this thesis for our research, but also its difficulties, which in our view stem from the computational hypothesis of the mind, of which Simon was one of the great promoters, i.e. the idea that all rational activity proceeds by the application of rules. *(4.4)* These difficulties lead us to develop a reflection on the difference between *technical rule* and *practical sense*. We start with a terminological study of the words *practical* and *technical* and expand it through the reading of ethical philosopher Alasdair MacIntyre and of sociologist Pierre Bourdieu. This inquiry allows us to identify two quite distinct issues concerning the relationships between practical knowledge, technical knowledge, and science.

5 Practical reason and industrial logic

(5.1) We explore further the notion of practical rationality. We leverage anthropological studies of work on the use of rules and procedures in technical environments, as well as *(5.2)* examples of conflicts between the rigid rules of information systems and the fluid functioning of organizations. *(5.3)* We argue that strict procedural logic, far from being natural to people, is characteristic only of industrial environments. This leads us to question the conceptual proximity between computing and industry, which is also historical (with Babbage); we find them both characterised by the same analytical and systematic approach to the problems to be solved. *(5.4)* This allows us to identify a typology of the *relative advantages* of effectiveness that computing brings, which are similar to those of industry, and which extend far beyond mere productivity gains. *(5.5)* We thus put forward the hypothesis that computing and industrial methods are not a kind of practical or technical knowledge that would be simply characterised by an increased care to the rigour or precision of their rules, but knowledge of a radically different form, which has often been described as "scientific". The problem then is to explain how this knowledge, if it is in such an *epistemic break* with ways by which people act, could be endowed with such an *effectiveness* ? *(5.6)* We begin our exploration of this difficult hypothesis by examining Ryle's particularly original conception of theoretical knowledge, which invites us to take the problematic concept of *systematicity* as our guiding thread.

Part III

Theory and practice

In this part we raise the question of the possibility of a "systematic" knowledge that would deal with problem solving, including if it be of a practical or technical nature. We approach this question by asking what transformations of the very concept of *knowledge* were required in order for this possibility to become *thinkable*.

6 Deliberation and system

We begin by delving into the paradoxical nature of such an idea by analysing Aristotle's distinction between scientific reason, which deals with the necessary principles of things, and deliberative reason, which deals with the contingent objects of the will. This *generic* distinction seems to deny the latter the possibility of expressing itself in the form of a deductive system (and thus, presumably, of being systematic). Our analysis focuses on technique (*τέχνη*). We study ***(6.1)*** first, the proximity of this kind of knowledge to science but also its differences, and then ***(6.2)*** to prudence (*φρόνησις*). This leads us to propose an original interpretation of the status of *τέχνη* in Aristotle. ***(6.3)*** The question then becomes under which conditions it could be systematic. We study a method of inquiry that Aristotle presents as "universal" in book I of *Prior Analytics*, traditionally known as *pons asinorum*. ***(6.4)***. Our conclusion provides a nuanced assessment of Aristotle's scepticism about the possibility of technical systems.

7 Machine and system in Modern Times

In this chapter we attempt to lay down some milestones as to the emergence, in history, of the conception of this possibility. ***(7.1)*** We begin by recalling the many transformations that the theory / practice pair underwent from the early Middle Ages onwards, and which attenuated its opposition. We then turn our attention to a great effort aiming at the rationalisation of practices characteristic of Modern Times through works called *reductions in art*, i.e. attempts at a reasoned description of a craft or art, making explicit their rules and underlying rationality. ***(7.2)*** We study the limits of the systematic forms available to these works. We contrast

them with the form of geometry, *(7.3)* which is the bedrock of the military engineer's practice, allowing him to envisage a *total* conception of his work, removing (ideally) all need for initiative and decision in its execution. *(7.4)* We are particularly interested in the singular status of machines, both objects of engineering design and scientific explanation of natural phenomena. *(7.5)* We put this status in perspective of the close relationship mechanics had with geometry in Modern Times. *(7.6)* We conclude this chapter by examining the emergence of the notion of *system* and its critique in the 18$^{\text{th}}$ century.

8 The rules of judgment and action

This chapter is devoted to establish a dialogue between Kant and Aristotle on the status of technical knowledge, which the German philosopher presents, in his famous introduction to the *Critique of the Faculty of Judgement*, as a "corollary" of theoretical knowledge. *(8.1)* We analyse the meaning of such an assertion by laying out the central role played by the notion of *rule* in the *Critique of Pure Reason*. *(8.2)* We present Aristotelian-inspired objections to this reduction of practical rationality to theoretical understanding, first Anscombe's, then *(8.3)*a modal argument which contrasts the description of the situation with the prescription of the option, and finally the argument of recent philosophers of technology, which insists on the particular forms and purposes of technical knowledge. *(8.4)* We claim that these arguments, despite their force, fail to challenge the Kantian position. However, the latter presents a new difficulty for our research, in that Kant confines the notion of *system* to its architectonic function in the unification of knowledge. This interpretation fails to explain, in effect, what positive role systems could play in the very inquiries of understanding, and thus what might be meant by the *systematic* approach to problem solving that could be at the root of the methods of industry and computer science.

Part IV

Problem solving

In this part, on the basis of what we have learned in our previous two parts, we seek to identify what might be a *logic of inquiry* that would show the possibility

of its systematic form.

9 Constructing and proving

We follow the thread of mathematics, which appeared to us (particularly geometry) as the locus par excellence of systematic problem solving. ***(9.1)*** We are first interested in the notion of problem itself, and particularly in the classical duality between problems of *proving* a proposition and problems of *finding* an object (a number, a function, a figure, etc.). We present the solution offered by Per Martin-Löf's constructive type theory to unify such a distinction. In our view, it allows to think afresh about the notion of *problem*, making visible new distinctions in its core. ***(9.2)*** We test the applicability of such a theory to Euclidean geometry. In particular we test if it allows a new interpretation to emerge on the ancient debate, reported by Proclus, concerning the primacy of *theorems* or of *problems*. ***(9.3)*** After studying the attempts already made on this question, we sketch an original solution that helps to illustrate the potential complexity of problem structures. ***(9.4)*** We are then finally able to propose a characterisation of systematic problem solving according to three principles. ***(9.5)*** Our research is not complete, however, because mathematics does not explain the relationship between inquiry itself and the systematic problem solutions it arrives at. In the terms of our subject, it is as if mathematicians gave us the *programmes*, but told us nothing about the *programming* itself.

10 The theory of inquiry

We address this question to John Dewey, who made *inquiry* the core of his philosophy of knowledge. ***(10.1)*** We briefly recall the central concepts of this theory, in particular those of meaning and language, ***(10.2)*** then those of judgement and experience, before presenting his conception of problem solving. This allows us to set out his interpretation ***(10.3)*** of the nature of mathematics, ***(10.4)*** of scientific theories, as well as ***(10.5)*** of institutional and technical systems. We argue that the constructive theory of types provides a convincing complement to this epistemology, and that together they allow us to think about both the continuity and the rupture that exists between practical and technical inquiries on the one hand, and systematic inquiries on the other. ***(10.6)*** We present a general summary of the epistemological achievements of the last three parts.

Part V

Machines and systems

The previous three parts have enabled us to identify a minimal epistemological framework around the notion of *systematic problem solving*. In this final part, we complete our elucidation of the nature of programming, in particular through a reflection on the notion of *machine*, as well as the reasons for its effectiveness in our human world.

11 The programming of machines

(11.1) We analyse which relationship exists between programming and systematic inquiry, taking as our guiding thread the notion of *effective procedure* in computability theory. ***(11.2)*** We show that this fundamental relationship makes it possible to lay the foundations of an epistemology of computer science, in its main divisions (algorithmics and formal methods) as well as in its theoretical and practical dimensions. ***(11.3)*** We now question the necessary link that must exist between this concept of programming and that of *machine*, by questioning the mathematicians' repeated demand that the human computer behave *mechanically*. This leads us to propose a new definition of the notion of machine, as any *dynamic phenomenon whose interaction with its environment can seemingly be described by rules*. ***(11.4)*** We show the logical possibility of constructing a computer through what we call a *transmission control* mechanism, ***(11.4)*** and we explain in what sense some of these machines can be said to be *universal*. ***(11.5)*** We deepen the notion of mechanism through the confrontation of the theses of *pan-computationalism*, a metaphysical doctrine, and *new mechanism*, an epistemological doctrine. ***(11.6)*** We sketch the ways in which programming transforms the systematic inquiries of mathematics, science and engineering.

12 To make the rules effective

Finally, we explore the effectiveness of computers outside the field of inquiry, within human practices themselves. ***(12.1)*** We clarify the concept of *interactivity*, which is central to theoretical computer science ***(12.2)*** and show to what extent an interactive procedure is capable of creating a *coupling* between machine and

situation that makes programmed rules *directly effective*. ***(12.3)*** This possibility infiltrates the functioning of our objective systems of rules, institutional and technical, and thus makes them *dynamic*. ***(12.4)*** We show how this new type of situation, although *systematic*, carries the risk of a new disorder.

13 Conclusion

(13.1) We take up all the questions raised by this work and provide a succinct response. However, two important questions remain open: the status of statistical machine learning and the nature of the machines' effectiveness. ***(13.2)*** We mobilise our epistemological framework to analyse statistical machine learning and suggest that it extends the domain of systematicity to forms of problem hitherto inaccessible to programming. ***(13.3)*** We clarify the sense in which the *effectiveness of machines* is to be understood, and how it derives entirely from the *effectiveness of knowledge* itself. ***(13.4)*** Finally, we show how our work may shed light on the ethical and political debates linked to the upheavals generated by the digital era.

Keywords

Aristotle – Kant – Dewey – Ryle – Jackson (Michael) – Simon (Herbert) – Martin-Löf– Machine – Rule – Procedure – System – Inquiry – Practice – Technology – Program – Computing

Remerciements

Dominique Pradelle, il y a quelques années, a soutenu ce projet et m'a orienté vers Pierre Wagner. Celui-ci m'a accompagné patiemment et rigoureusement tout au long de ce périple. Ses lectures attentives, ses commentaires, ses réorientations discrètes mais efficaces, ses encouragements réguliers m'ont été précieux.

J'ai retrouvé avec bonheur l'IHPST, où j'avais fait mon DEA il y trente ans, sous la direction de Michel Fichant. J'ai eu la chance pendant ma thèse d'assister à sa très belle modernisation. Les échanges avec Alberto Naibo, Marco Panza, et Jean Fichot m'ont stimulé et éclairé. J'ai beaucoup ferraillé avec David Waszek au sujet de Herbert Simon et avec Matias Osta Vélez sur les sciences cognitives. Adrien Champougny et Perceval Pillon ont toujours répondu avec bienveillance et patience à mes questions naïves de logique. Dans le cadre de l'IHPST, l'ANR GOA (ANR-20-CE27-0004) m'a permis de travailler de manière suivie avec plusieurs de ces personnes, ainsi qu'avec Walter Dean et Philippos Papayannopoulos. L'amitié qui soude la communauté des doctorants m'a rajeuni de vingt ans, et en particulier celle de ses figures tutélaires, Marie Michon et Caroline Anglereaux.

Giuseppe Primiero et Liesbeth De Mol m'ont accueilli avec beaucoup d'ouverture d'esprit dans la communauté HAPOC, dès les début de mon travail, alors que je ne savais pas encore grand-chose, puis dans l'ANR PROGRAMme (ANR-17-CE38-0003-01) qui est rapidement devenue une deuxième famille, notamment grâce à nos inoubliables soirées de Bertinoro. Beaucoup d'idées présentées ici ont mûri avec ce groupe, et cette thèse peut être considérée comme un « compagnon » de l'ouvrage collectif qui doit être publié l'année prochaine. J'ai tant appris sur l'histoire et la philosophie de la programmation auprès de Nicola Angius, Troy Astarte, Maarten Bullynck, Selmer Bringsjord, Felice Cardone, Martin

Carlé, Marie-Jo Durand-Richard, Jean-Baptiste Joinet, Cliff Jones, Jean Lassègue, Baptiste Mélès, Simone Martini, Pierre Mounier-Kuhn, Elisabetta Mori, Julian Rohruber, David Schmudde, Máté Szabó, Ray Turner, Nick Wiggershaus et bien d'autres. Je mets à part Maël Pégny, qui a été mon premier interlocuteur tout au long de ces années.

Sjoerd Zwart, Pierre Steiner, Matthias Girel, David Hildebrand, Brice Halimi, David Rabouin, Violetta Lonati, Natacha Gaudillère-Jami, m'ont fait l'amitié d'échanges approfondis et d'éclairages sur des sujets auxquels j'étais étranger. Liliane Hilaire-Pérez et Stefanie Buchenau m'ont accueilli dans leurs séminaires de recherche.

François Ruisz et Jean-Claude Guyard, de la société Cap Gemini, Alexandre Aractingi et François Candelon de la société BCG, ont consacré un temps précieux à partager avec moi les dernières tendances de la programmation professionnelle.

Indications pour la lecture

La table des matières se situe à la fin du document. Dans le document électronique pdf, tous les renvois (à la bibliographie, au glossaire, vers des sites Internet, depuis les tables analytiques, et enfin les renvois internes au texte) sont actifs et peuvent donc être cliqués, si le logiciel de lecture offre cette fonctionnalité.

Au-delà des divisions en parties et chapitres, les unités de composition du document sont la section (4 à 6 par chapitre, avec numérotation dépendante), et le développement (3 à 5 par section, avec numération continue sur l'ensemble du document, introduit par le sigle §). Chaque développement correspond à peu près à argument ou à une idée. La section et le développement en cours sont indiqués dans l'en-tête de la page impaire.

Un glossaire technique est fourni pour la lectrice non informaticienne. Les termes renseignés sont marqués dans le texte par une astérisque*. Les noms de technologies (logiciels, langages, standards,...) et de sociétés commerciales sont marqués par cette graphie spéciale. Cela permet d'indiquer que des droits (*copyright* ou *trademark*) pourraient leur être associés.

Les références bibliographiques sont indiquées en général par un renvoi de forme (Auteur, année de publication) ou Auteur (année de publication) vers la bibliographie finale. Cependant, pour les auteurs classiques, l'ouvrage consulté est plutôt précisé par le nom de l'éditeur ou du traducteur. Sa référence complète est donnée, dans la bibliographie, sous la clé de l'auteur.

Les traductions sont en général les nôtres, sauf mention contraire.

Chapitre 1

Introduction

1.1 L'effectivité de l'informatique

§ 1. La question initiale

L'étonnement initial qui a motivé ce travail porte sur la présence généralisée de l'informatique dans notre monde humain. Ce que nous entendons par cette expression se constate aisément. Les ordinateurs sont dans notre poche, se cachent derrière tous nos écrans, et sont intermédiaires à la satisfaction de la plupart de nos besoins et de nos loisirs : achats, transports, communications, relations sociales, jeux, lectures, spectacles. Les ordinateurs sont également centraux à l'opération des institutions et des organisations humaines, qu'il s'agisse d'entreprises ou d'administrations. Le système financier global est de nature entièrement informatique, si bien que de nouvelles monnaies, indépendantes du système bancaire, sont en train de voir le jour. Les ordinateurs ont pris le contrôle des usines et de chaînes logistiques, et sont embarqués dans toutes sortes d'artefacts robotiques ou d'objets usuels qu'ils rendent réactifs à leur environnement, comme un parcmètre dans la rue ou une valve dans un réseau hydraulique. Ils pilotent les avions et prennent progressivement le contrôle de la voiture. Ils transforment, pour la même raison, les armements et la nature de la guerre. Enfin, ils transforment la science elle-même dont les progrès sont aujourd'hui, dans de nombreux domaines, entièrement dépendants d'instruments de mesure, de simulations et de calculs impraticables sans une puissance informatique considérable. Le progrès de larges entreprises de connaissance empirique, comme l'analyse du ciel, des interactions de particules, ou du génome humain est devenu entièrement dépendant de leur in-

formatisation. L'informatique investit les méthodes et les idées même de ce qu'est le savoir : la notion de preuve en mathématiques fait l'objet d'âpres discussions depuis que des mathématiciens ont proposé des preuves exécutables par ordinateur. L'ampleur et la profondeur de ces changements a conduit Gérard Berry, titulaire de la chaire d'informatique au Collège de France, à parler d'une « hyperpuissance de l'informatique[1] ».

De tels constats n'ont rien de nouveau. Ils sont formulés régulièrement depuis que le grand public s'est passionné pour ces « cerveaux électroniques » apparus dans après la seconde guerre mondiale. La formulation de cette fascination change au gré des évolutions technologiques : dans les années 1960, on insiste sur l'intelligence artificielle, dans les années 1970, sur les robots, les systèmes experts et les bases de données, dans les années 1980 sur l'informatique personnelle puis sur le « multimédia ». À partir de 1995, ce sont les « autoroutes de l'information » puis l'Internet qui fascinent les esprits, et après 2005, l'Internet mobile et les réseaux sociaux. Depuis 2015, l'intelligence artificielle et la robotique sont à nouveau sur les devants de la scène, bien qu'on désigne par là de tout autres technologies que cinquante ans auparavant. Cependant, au-delà de ces différences de surface, il y a le même étonnement constant, admiratif ou inquiet, devant les progrès de cette technologie dont on ne perçoit pas encore les bornes.

La banalité de cet étonnement sans cesse renouvelé peut faire douter de son authenticité, ou tout au moins de son intérêt pour la philosophie. L'historien des sciences et des techniques Paul Mahoney a très tôt dénoncé la rhétorique convenue de la révolution numérique, centrée sur la machine et empreinte de déterminisme technologique[2]. On peut en effet remarquer que l'admiration du grand public s'est également exprimée pour quantité d'autres inventions de la modernité, qu'il s'agisse du train, de l'avion, de l'automobile, du télégraphe et du téléphone, de la machine à vapeur, de l'électricité, de l'énergie nucléaire, ou encore, dans un autre registre, de l'enregistrement sonore, de la radio, de la photo, du cinéma, de la télévision. On a ainsi parlé de l'« âge de l'automobile » aussi bien que de l'« ère atomique » ou de la « société de la télévision », presque tout autant qu'on parle aujourd'hui de la « révolution numérique ». Chacune de ces inventions majeures a transformé les modes de vie, l'économie, la société, et la guerre. Ces transformations peuvent certainement motiver une réflexion philosophique générale sur la

1. (Berry 2017).
2. (Mahoney 2011, p. 55-60). Voir également l'introduction de Thomas Haigh à cet ouvrage.

technique, mais sans doute pas une « philosophie de l'automobile » ou de l'électricité.

La suite de cette introduction est consacrée à expliciter l'intérêt pour la philosophie de cet étonnement initial, en montrant son inscription dans des problématiques plus larges.

Dans cette section, nous montrons d'abord que cet étonnement peut être formulé par deux questions, l'une concernant la polyvalence des ordinateurs (§ 2), et l'autre leur effectivité (§ 3). Une objection est finalement considérée : ces questions ne trouvent-elles pas une réponse technique très simple (§ 4) ?

Les sections 1.2 et 1.3 examinent diverses réponses à ces questions, dont le point commun est de lier de manière étroite informatique et connaissance. Tout en montrant leur insuffisance, nous plaçons l'hypothèse qui les guide au centre de ce travail, dont le titre général est donc : « la programmation est-elle une forme de connaissance ? »

La section 1.4 développe cette question et montre son caractère problématique : elle ne peut trouver une solution simple à l'aide des catégories épistémiques habituelles. Elle requiert de s'inscrire dans des problèmes plus généraux concernant la nature de la connaissance, notamment ceux liés aux rapports entre théorie et pratique. Ce cadrage large nous permet de situer, en section 1.5, notre problématique relativement à plusieurs questions actuelles en philosophie de la connaissance. La section 1.6, enfin, situe notre travail vis-à-vis de problèmes apparentés de philosophie de l'informatique et précise son plan et son approche.

Ainsi notre problématique, partant d'une question initiale au sujet de l'effectivité des ordinateurs dans notre monde, s'élargit progressivement, comme par cercles concentriques, en direction de questions plus globales concernant la connaissance dans son ensemble.

La longueur de ce parcours introductif s'explique par les rencontres, en chemin, de nombreuses explications et interprétations des faits qui nous interrogent, issues non seulement de la philosophie, mais également de l'informatique elle-même, de l'anthropologie ou des sciences cognitives. L'examen de ces thèses importantes qui ont nourri notre réflexion, mais que nous ne pourrons souvent pas mieux développer dans la suite de ce travail, est nécessaire pour construire notre questionnement.

§ 2. Polyvalence des ordinateurs

La singularité de l'ordinateur est manifeste lorsqu'on le compare aux autres grandes inventions de la révolution industrielle citées plus haut. S'il est aisé de déterminer le domaine d'utilité de chacune – la communication pour le téléphone, le transport pour la voiture, l'énergie pour l'électricité – que dire de celle de l'ordinateur ? Son nom américain (*computer*) se réfère au calcul, les termes français *ordinateur* et *informatique* à celles d'ordre et d'information. Ces notions n'ont pas de liaison évidente entre elles, et semblent refléter une multiplicité de perspectives possibles sur la fonction des ordinateurs. La référence à la notion d'information rapproche par ailleurs l'informatique des télécommunications, mais l'ordinateur fait apparemment bien plus de choses que transmettre des messages. Il semble opérer sur un autre plan que toutes ces inventions.

Si la voiture, l'électricité ou le téléphone ont investi et transformé des activités humaines aussi diverses que l'industrie, le commerce, le loisir, c'est parce que le transport, l'énergie et la communication sont des fonctions si essentielles que leur transformation ne pouvait manquer d'ouvrir de nouvelles perspectives à ces activités. Peut-on en dire autant de l'informatique ? Si on dit qu'elle automatise le calcul, cela explique-t-il l'ampleur des changements constatés ? Cela n'est pas évident, comme le note l'informaticien Anatol Holt en relatant sa première découverte des ordinateurs, en 1952 :

> J'ai été frappé par une énigme. Comme tant d'autres technologies, l'ordinateur était un enfant de la guerre. Dans ce contexte, il était censé aider à effectuer des calculs extrêmement fastidieux, qui conduisaient à des chiffres, ce qui conduisait à plus de puissance de feu contre l'ennemi. Il était censé calculer, c'est pourquoi on l'appelait un ordinateur[3]. Cependant, très peu de clients potentiels de cette nouvelle machine avaient des problèmes de ce type. Ils l'utilisaient plutôt pour trier des données, contrôler le trafic ferroviaire, émettre des feuilles de paie, etc. Dans mon jeune esprit, cela soulevait une question brûlante : si l'ordinateur ne sert pas vraiment – ou principalement – à calculer, alors, à quoi sert-il[4] ?

On peut arguer que les activités de calcul sont, de manière invisible, sous-jacentes à de très nombreuses activités humaines, comme le sont la communication et le transport. Mais peut-on vraiment expliquer le succès des réseaux sociaux par le

3. *Computer* en anglais veut dire *calculateur*. Nous éviterons autant que possible le terme de *computation* et ses dérivés, au profit de *calcul*, sauf dans les expressions où l'usage l'a consacré, comme dans l'*hypothèse computationnelle de l'esprit* que nous examinons plus loin.
4. (Holt 1997, p. 7).

fait que les relations humaines sont des calculs ? Cela ne semble pas très crédible, même si on comprend cela de manière cynique. On peut également proposer d'entendre *calcul* en un sens beaucoup plus large que celui qui l'associe couramment à l'arithmétique, comme « traitement de symboles selon des règles » – mais, si cela peut éclairer quelques exemples d'applications données par Holt, cela ne semble pas s'appliquer de manière évidente à bien d'autres.

Une autre réponse courante à cette question est d'affirmer que les ordinateurs sont des machines « de traitement de l'information ». Nous l'examinerons plus loin dans cette introduction mais, pour le dire rapidement, il nous semble qu'on ne fait là qu'enfouir le mystère dans les méandres des significations d'un mot passe-partout.

Nous appellerons *polyvalence* cette capacité qu'ont les ordinateurs de pouvoir apparemment s'adapter à toutes sortes de besoins. Nous évitons le terme *universalité* car nous ne présupposons pas qu'ils puissent s'adapter à *tout* besoin.

Cette polyvalence, qui étonne Holt, peut s'expliquer d'un autre point de vue, qui n'exige pas de résoudre d'emblée la question de leur fonction primaire. Les ordinateurs se distinguent des autres inventions en ce qu'ils sont *programmés* pour répondre aux différents besoins qui les sollicitent, et ils sont donc des artefacts essentiellement *incomplets* sans les logiciels ou applications qui déterminent leur fonctionnement particulier, dans tel ou tel contexte. Les ordinateurs sont avant tout des machines *programmables*, et c'est par ce biais qu'ils sont polyvalents, par leur adaptabilité à la résolution de toute sorte de problème.

C'est ce fait essentiel et aisément observable de la programmation qui doit donc nous interroger. Comment se fait-il que tant d'activités humaines soient « programmables » ? Est-ce parce qu'elles ont essentiellement la forme de plans d'action, de procédures, que les informaticiens ne feraient que retranscrire dans un langage compréhensible par l'ordinateur ? Y aurait-il ainsi un langage universel capable de décrire toute action ? Et y a-t-il des limites à cette retranscription ?

§ 3. Effectivité des ordinateurs

Cette question en appelle une autre. Qu'on parvienne à décrire un grand nombre d'activités humaines dans un langage universel des procédures est une chose, c'en est une autre que ce langage soit effectif dans la réalité, sans l'intermédiaire d'un agent humain qui l'interprète en agissant. On parle souvent de la fonction perlocutoire du langage pour signifier les effets qu'il produit chez

l'interlocuteur[5], et ces notions ont parfois été proposées pour décrire l'effectivité des ordinateurs[6]. Cependant, c'est de tout autre chose qu'il s'agit ici, puisque les effets en question ne concernent pas des personnes, mais des artefacts matériels.

Comment est-il en effet concevable qu'un ensemble de formules symboliques, ou une suite de 0 et de 1 – ce à quoi se réduit ultimement le programme d'un ordinateur – puisse s'animer et s'exécuter de manière à accomplir des fonctions signifiantes dans le monde, comme piloter un avion, ou même simplement déterminer l'affichage d'un écran ? La possibilité d'une telle effectivité des significations – la possibilité de l'ordinateur tout court – n'est pas claire. Le philosophe des sciences Gualtiero Piccinini expose bien la difficulté :

> Si vous pensez que la théorie de la calculabilité concerne des objets abstraits et qu'il existe de telles choses – c'est-à-dire si vous êtes platonicien – vous devez vous demander : que faut-il à un système physique concret donné pour qu'il implémente un objet calculatoire abstrait donné (par opposition à un autre objet abstrait, ou aucun) ? […] Si vous pensez qu'il n'y a pas d'objets abstraits, vous devez vous demander : que faut-il à un système physique concret donné pour qu'il satisfasse une description calculatoire abstraite donnée (par opposition à une autre, ou aucune)[7] ?

Ainsi par exemple, l'informaticien Tim Colburn, dans sa réflexion sur la nature de l'informatique, se résigne à faire appel au schéma leibnizien de l'harmonie préétablie entre l'âme et le corps pour expliquer l'accord merveilleux entre les représentations abstraites des programmeurs et le comportement matériel de l'ordinateur – le Dieu horloger étant ici remplacé par la longue chaîne anonyme des informaticiens ayant développé les compilateurs[*8], les systèmes d'exploitation[*], et les circuits intégrés qui établissent cette liaison [9]. Cependant, on ne voit pas clairement comment une chaîne de traducteurs, à partir d'un texte initial, produirait autre chose que des traductions, c'est-à-dire d'autres textes. Que font donc ces programmeurs pour établir une telle harmonie, s'ils ne font eux-mêmes que manipuler des significations ?

La seconde question concernant l'effectivité des ordinateurs est donc tout simplement comment est *possible* cette effectivité qui réalise des significations dans le monde, sans l'intermédiaire direct d'un interprétant humain.

5. (Austin [1955] 1975, p. 101).
6. Voir plus loin, au développement § 195, l'idée de l'informatique comme *pragmatique* des mathématiques.
7. (Piccinini 2015, p. 7).
8. Le symbole (*) marque un terme technique explicité dans notre Glossaire.
9. (Colburn 2000, p. 208).

Il semble qu'on ait affaire ici à un question ontologique, typique d'un dualisme qui doit expliquer comment les deux règnes qu'il distingue peuvent interagir, ici les significations du programme et l'exécution matérielle de celui-ci sur des machines.

Ce n'est pas de cette manière que nous souhaitons envisager la question, car elle obligerait à expliciter ce que nous entendons par *monde, réalité, etc.* et donc à nous placer d'emblée sous les auspices d'une philosophie particulière, qu'elle soit dualiste ou moniste, réaliste ou sceptique, etc. Il est peu probable que notre question directrice, qui concerne un phénomène empirique et historique – les ordinateurs et leur rôle dans le monde humain – doive recevoir des réponses différentes selon le cadre métaphysique dans lequel on se place.

Le choix du terme *effectivité* reflète notre volonté de nous tenir à l'écart de ces questions ontologiques. Le mot est peu courant en français. Il n'est pas attesté par le Robert, et le Larousse l'admet seulement comme substantivation d' *effectif*. Selon ces deux dictionnaires, cet adjectif a d'abord le sens très générique de « réel, tangible, positif ». Le Larousse marque également deux autres sens d'*effectif*.

Il y a d'abord celui de « prendre effet » comme dans : « *L'armistice sera effectif à 11 heures.* ». C'est le sens principal qui nous intéresse ici. Les ordinateurs aident *effectivement* les personnes humaines à résoudre des problèmes auxquels elles sont confrontés, comme traiter des factures, contrôler l'accès à un bâtiment, piloter une sonde spatiale. Ils permettent à leurs demandes et à leurs spécifications de « prendre effet ».

Ce sens est bien marqué en anglais, où on oppose souvent *effectiveness* à *efficiency* (efficacité). L'effectivité consiste à parvenir à ses fins, quelles que soient les méthodes utilisées, tandis que l'efficacité mesure le rendement d'une méthode donnée – qui peut se révéler finalement « ineffective » si ses conditions de réalisation ont par exemple disparu au cours des travaux. On peut très *efficacement* pomper dans le vide.

Nous avons également choisi *effectivité* afin d'éviter le terme de *puissance*, souvent utilisé à propos de l'informatique, voire celui d'« hyperpuissance » proposé par Gérard Berry. Ces mots évoquent un « règne » ou une domination – des connotations sur lesquelles nous reviendrons tout à l'heure[10]. L'*effectivité* est une notion plus « modeste ». Contrairement à ces termes, elle est *locale* et circonscrite au problème dont elle s'occupe. Elle désigne la propriété d'un objet servant de manière essentielle à effectuer *ce qui est demandé*.

10. Voir plus loin, § 20.

L'intérêt de cette formulation est qu'elle ne présuppose pas que ces problèmes soient les bons, qu'ils augmentent le pouvoir des êtres humains sur le monde, le bien-être individuel ou l'utilité générale. On peut utiliser les ordinateurs pour résoudre de mauvais problèmes, ou des problèmes insignifiants. Pis encore, il se peut que les ordinateurs induisent les personnes à se poser des problèmes qu'ils ne se seraient jamais posés sans leur présence, ou à les poser d'une manière qui les rendent indispensables, etc. Nous y revenons tout à l'heure[11].

Il est important ici de clarifier en quel sens nous entendons ici le terme *problème*. Cette clarification n'est que préliminaire, puisque nous n'éluciderons entièrement ce qu'il signifie qu'en quatrième partie de ce travail. Cependant, il faut préciser d'ores et déjà que nous ne limitons pas le champ de cette notion aux problèmes intellectuels auxquels s'est d'abord intéressée l'informatique : problèmes logiques et mathématiques, jeux, énigmes ludiques, etc. *Problème* doit être pris dans un sens bien plus général, c'est-à-dire celui de la représentation d'une situation d'insatisfaction, d'une *demande* qu'on saurait décrire. Néanmoins, dans le cadre de ce travail, nous ne prenons pas le terme dans son acception la plus grande, et ne considérons pas les larges classes de problèmes qu'on appelle *philosophiques, éthiques, esthétiques, etc.* – ces problèmes où les notions même de *résolution* et d'*effectivité*, au sens où nous les entendons ici, sont généralement considérées (et à bon droit) comme impertinentes.

L'effectivité comme « prise d'effet » est donc relative à une demande, à une prescription, à un problème spécifique, et elle leur emprunte leur statut ontique. La réalité du comptable peut être différente de celle de l'astronaute et de celle du mathématicien, et chacun d'eux exprime ses problèmes dans un langage spécialisé. La question de l'effectivité des significations se pose à ce niveau local, elle concerne la manière par laquelle un état de choses désiré, *tel qu'il est décrit dans le langage pertinent*, peut être réalisé de manière à rendre valide cette description.

Cette interprétation est confirmée par le troisième sens d'*effectif* indiqué par le Larousse, qui atteste son usage technique en informatique : « Se dit d'une méthode qui, appliquée à la résolution d'un problème, en donne la solution au bout d'un nombre fini d'étapes (Une opération est dite effective s'il existe un algorithme permettant de la calculer)[12] ». Nous aurons amplement l'occasion de revenir sur cette signification dans le cours de ce travail ; son articulation avec le sens plus général

11. Voir plus loin, § 20.
12. Dictionnaire *Larousse* en ligne, larousse.fr.

d'*effectif* (« prendre effet ») n'apparaîtra cependant qu'en conclusion.

§ 4. La réponse technique à ce questionnement

On pourrait objecter que l'on se pose ici de faux problèmes. L'effectivité des ordinateurs est une réalité technique, et elle doit donc être expliquée techniquement. Pour expliquer la diversité des applications de l'informatique, le mieux n'est-il pas tout simplement d'étudier les programmes qui sont sous-jacents à chacune, ligne à ligne et pas à pas, ou encore les circuits intégrés qui les implémentent ?

Chaque phénomène informatique individuel, qu'il s'agisse du contrôle d'un bras robotique, de celui d'un serveur vocal interactif, ou d'une simulation scientifique, est en effet parfaitement explicable, du moment qu'on peut accéder à l'organisation logique des programmes qui le pilotent et au code individuel de ceux-ci. Il n'y a même rien de plus explicable que cela, *en principe*, si on y consacre le temps nécessaire. De simples raisonnements logiques permettent de rendre compte de ce qui se passe, pourvu que les termes employés par le code des programmes soient définis, et que les hypothèses concernant le comportement des composants matériels de l'ordinateur soient spécifiés. Certes, la vitesse et la quantité des opérations effectuées rendent de telles explications impossibles pratiquement ; mais la possibilité de les réaliser sur n'importe quelle partie du système proportionnée à nos capacités doit nous convaincre qu'il n'y a nul mystère derrière tout cela.

En d'autres termes, nos questions ne se dissipent-elles pas sous le regard avisé du programmeur, qui est capable d'analyser et de comprendre les programmes informatiques, de la même manière qu'un ingénieur est capable d'expliquer pourquoi un pont tient debout ? L'étonnement du grand public serait ainsi celui de néophytes qui, ne pouvant expliquer des phénomènes du fait d'un bagage technique insuffisant, les rapportent de ce fait à la magie.

Il y a néanmoins deux problèmes à cette fin de non-recevoir. D'abord, posséder l'explication d'un programme ne suffit pas toujours à dissiper l'étonnement concernant ses résultats. De manière régulière, les programmeurs ont fait part de leur *propre* étonnement ou fascination vis-à-vis de leur activité, comme si celle-ci était dotée d'une épaisseur propre qui résistait aux simples explications de la raison – nous revenons sur ce point tout à l'heure[13].

Ensuite, il n'est pas évident de circonscrire le « bagage technique » de la programmation. S'agit-il de comprendre comment fonctionnent les ordinateurs,

13. Voir plus loin § 10.

d'avoir des rudiments d'électronique ? ou faut-il connaître au moins un langage de programmation ? – Ces savoirs, s'ils sont utiles, ne sont pas essentiels. Comme nous l'avons dit plus haut, la simple logique est souvent suffisante pour comprendre le fonctionnement d'un programme, du moment qu'on se laisse expliquer le sens des termes qu'il utilise – nous en donnons un exemple plus loin[14]. On a ainsi avancé que la programmation n'était pas une discipline technique, mais mathématique ou logique[15].

La logique n'est cependant pas suffisante, car programmer ne requiert pas seulement de coder, mais également de décrire un problème concret en termes informatiques, et c'est cela qui rend le codage possible. Or traduire exige de comprendre, et l'une des difficultés majeures que rencontre tout programmeur généraliste est de pénétrer la nature très diverse des problèmes qui peuvent lui être proposés, d'en saisir la logique interne en quelque sorte, de manière à pouvoir la transcrire informatiquement. Une telle ouverture à toute la diversité du monde humain n'est pas très courante. On la trouve également chez les avocats et, si on la restreint au monde économique, chez les financiers, les comptables et les consultants. Toutes ces activités de *services professionnels*, comme on les dénomme, développent des méthodes génériques d'action qui se moulent, tant bien que mal, à des réalités concrètes extrêmement différentes : commerce, industrie, divertissement, construction, services publics, etc.

Cependant l'informaticien se distingue de toutes ces professions par au moins deux aspects : d'abord, en ce qu'il construit ou modifie des artefacts matériels (les ordinateurs) pour rendre ses services, et d'autre part, en ce qu'il s'adresse également aux scientifiques eux-mêmes, et cela non pour les aider à résoudre des problèmes externes à leur activité (comme trouver un financement ou établir un contrat) mais bien leurs problèmes les plus propres, par le biais de logiciels de calcul ou de simulation par exemple.

Ainsi, il n'est pas aisé de déterminer ce qui manque à l'homme de la rue pour que se dissipe son étonnement face à l'effectivité générale des ordinateurs. Il ne semble pas s'agir essentiellement d'un bagage technique, ni d'un savoir mathématique, ni encore de méthodes professionnelles, même si tous ces ingrédients sont utiles à la formation d'un programmeur. La preuve en est qu'on peut commencer

14. Voir plus loin § 7.
15. Voir plus loin § 12.

à programmer très jeune, dès l'âge de 7 ans environ[16]. Aussi Berry, lors de sa leçon inaugurale au Collège de France, ne rapporte-t-il pas cet étonnement à des lacunes concernant des savoirs positifs concrets, mais plus généralement à des « schémas mentaux inadaptés » :

> Si la locution *monde numérique* s'entend partout, ses fondements restent largement ignorés du public, qui semble en permanence surpris par les innovations techniques et les transformations sociales associées. Or, au moins sur le plan technique, l'évolution est largement prévisible, et il n'y a pas de raison d'être surpris par du prévisible. La surprise permanente est plutôt le signe d'un schéma mental mal adapté, ce qui n'est pas étonnant car l'information synthétique est encore pauvre dans ce domaine qui ne repose pas sur des bases enseignées classiquement. L'ambition de cette leçon inaugurale est d'aider à construire un schéma mental approprié, autrement dit un *bon sens informatique*, en expliquant pourquoi le monde devient numérique, comment les transformations correspondantes se passent, et quels concepts et outils elles utilisent[17].

L'expression de « bon sens informatique » est d'une certaine manière un oxymore : car elle indique que le bon sens simple, celui que possède justement l'homme de la rue, n'est pas suffisant pour appréhender le phénomène informatique. Berry semble avancer par là, peut-être malgré lui, que l'informatique n'est pas seulement une question technique, mais qu'elle requiert une *réforme de l'entendement* – ce qui peut légitimement susciter une surprise encore plus grande que celle dont nous sommes partis.

1.2 L'informatique comme traitement de l'information

Que propose donc Gérard Berry pour adapter nos « schémas mentaux » à ce « bon sens informatique » ? Quel est, en particulier, le langage universel capable de décrire toute action, que nous cherchions tout à l'heure ?

Sa réponse est claire : ce langage est celui de l'information. Le premier pilier sur lequel repose l'informatique est « l'idée de représenter et de manipuler de façon homogène toute information quelle que soit sa nature, rompant l'identification ancestrale entre type d'information et support physique ». Cette section est consacrée à l'exposition et à la critique de cette thèse. Nous exposons d'abord (§ 5)

16. (I. R. Berson et M. J. Berson 2010, p. 50).
17. (Berry 2008, p. 16).

d'une manière générale la thèse de Berry, et nous l'illustrons ensuite (§ 6) de manière positive dans le domaine des échanges humains. Il semble bien, dans ce cas de figure, que la simple « dématérialisation » de l'information suffise à expliquer les larges bouleversements provoqués par l'informatique. Cependant, cette explication se révèle insuffisante dans d'autres cas de figure, car elle ne porte que sur la possibilité de *représenter* l'information de manière homogène, mais nullement sur celle de la manipuler, ce que Peter Denning, un autre informaticien, appelle le « paradoxe de l'information » (§ 7). L'examen de deux autres domaines d'intervention de l'informatique nous amène à conclure à la diversité des principes qui guident la programmation de ces types très différents d'opérations. Il ne semble pas y avoir de règle universelle de traduction, mais chaque fois un effort de programmation spécifique à réaliser, ce qui nous ramène à notre point de départ : qu'est-ce que programmer (§ 8) ?

§ 5. Le codage de l'information

Berry donne quelques exemples de la manière dont on peut coder l'information en représentations numériques homogènes :

> Le processus est trivial pour des informations numériques par construction, comme les sommes d'argent des financiers ou les valeurs physiques arrondies des scientifiques. Calcul scientifique et gestion financière ont donc constitué les deux premiers métiers des ordinateurs. Numériser des textes est à peine moins évident, car il suffit d'associer un code numérique distinct à chaque signe. Le traitement de texte a été le troisième métier des ordinateurs. Numériser des images et des sons est plus complexe. Pour une image *a priori* continue, il faut se contenter d'une grille de points discrets, où chaque point est une intensité lumineuse finie (par exemple de 0 à 255) en noir et blanc, ou bien un triplet d'intensités rouge-vert-bleu en couleurs. La transformation continu-discret se fait par un convertisseur analogique/digital, ici le capteur de lumière de l'appareil photo. Un convertisseur du même type transforme les sons en suites de nombres […] Les forces mécaniques se numérisent également à l'aide de capteurs adaptés. Réciproquement, un convertisseur digital/analogique transforme une suite de nombres en images, sons ou forces[18].

À travers les exemples de ce texte, Berry esquisse une histoire simplifiée de l'informatique, dont le développement aurait ainsi suivi la réalisation progressive de cette grande idée de la représentation homogène de toute information. De fait,

18. (Berry 2008, p. 22).

l'informatique de gestion et scientifique ont été prépondérantes jusqu'à la fin des années 1970, avant le développement de la bureautique dans les années 1980 (le « traitement de texte »), du multimedia dans les années 1990 (« les images et les sons »), et de systèmes embarqués et des robots à partir des années 2000 (« les forces mécaniques »).

Mais qu'entend Berry ici par *information* ? Ce terme, très ancien en philosophie, désignait jusqu'au Moyen Âge le processus par lequel une forme s'incorporait à la matière ; ce sens « ontologique » disparaît peu à peu dès les Temps modernes au profit d'un sens épistémique, qui désigne le processus par lequel on communique quelque chose de nouveau à quelqu'un[19]. De nos jours, il reste le sens principal de son usage courant. En philosophie cependant, on le nomme souvent *concept sémantique de l'information*, pour le distinguer de deux autres sens qu'il a acquis au XX^e^ siècle[20]. Depuis Claude Shannon, l'information est en effet l'objet d'une théorie mathématique, qui étudie la quantité d'incertitude associée à un processus aléatoire (le lien avec le sens principal d'information est qu'il y a moins d'information, par exemple, dans le résultat d'un jet de pièce à pile ou face que dans celui d'un lancement de dés) ; une autre théorie mathématique de l'information est celle de Kolmogorov, qui permet de mesurer plutôt la complexité intrinsèque d'une structure. L'information est aussi l'objet de théories physiques qui utilisent ces outils mathématiques pour décrire et expliquer le comportement chaotique de certains systèmes ; l'information est alors une grandeur physique objective, complémentaire de l'entropie, désignant le degré d'ordre interne d'un système[21]. L'information revient ici à son domaine « ontologique » originel, puisqu'elle désigne l'ordre imposé à la matière.

De nombreuses réflexions contemporaines ont tenté d'unifier ces différents concepts, physiques, mathématiques et épistémologiques ; certaines aboutissent au *pan-computationnalisme*, une famille de doctrines métaphysiques qui posent que l'information est la fabrique même de l'univers, et que nos processus de connaissance participent, d'une manière ou d'une autre, à des processus physiques fondamentaux. Nous examinerons plus loin dans cette réflexion une thèse séduisante, proposée par l'informaticien Gilles Dowek, qu'on pourrait rapprocher du pan-computationnalisme[22]. La philosophie de l'information de Luciano Floridi,

19. (Capurro 2009, p. 127-130).
20. Adriaans (2013), distingue, quant à lui six sens.
21. Voir (Uzan 2007) pour une mise au point utile sur ces notions.
22. Voir plus loin, § 210.

quoique récusant le pan-computationnalisme, élabore une métaphysique, une épistémologie et une éthique cohérentes en posant le concept d'information à leur fondement[23]. Sa discussion requerrait une étude à part. Mais d'une certaine manière, ces thèses, comme l'hypothèse computationnelle de l'esprit que nous étudions plus loin[24], répondent *de manière triviale* à notre question : car si le monde tout entier est traitement de l'information et calcul, alors il n'y a pas à s'étonner que les machines à calculer par excellence, les ordinateurs, soient d'une redoutable effectivité. Mais une fois cela affirmé, notre question reste également entière : quelles différences y a-t-il, précisément, entre notre manière de traiter l'information et celles de l'ordinateur, et comment expliquer qu'elles puissent si bien collaborer entre elles ?

Pour revenir à Gérard Berry, dans le texte en question c'est le concept sémantique de l'information (son sens courant, donc) qu'il a à l'esprit, puisqu'il en propose la définition suivante : il s'agit de données dotées d'une interprétation.

> L'information contenue dans une donnée dépend essentiellement du contexte d'interprétation. Si nous stockons dans une mémoire la valeur décimale 20 pour dénoter une température en degrés Celsius, cette valeur est une donnée, alors que « la température est de 20 degrés Celsius » est une information compréhensible en Europe mais pas aux États-Unis, où l'on parle en degrés Fahrenheit. Et une température de 20 degrés Fahrenheit (soit - 6,6 degrés Celsius) est froide pour nous mais parfaite pour un ours blanc. L'information est donc par essence une notion à géométrie variable[25].

Le terme central ici est celui d'*interprétation*, qui permet de transformer une donnée en information. 20 est une donnée (par exemple que je lis sur un thermomètre), mais 20°C et 20°F sont des informations (que j'infère en observant l'échelle du thermomètre). À son tour, 20°F est une donnée pour qui cherche à savoir si l'air extérieur est froid ou non : sur cet axe, un ours blanc en tirerait une information différente que nous. On voit donc ici que non seulement une même donnée peut donner lieu à deux informations différentes dans des contextes différents, mais également que la distinction donnée / information est relative, puisque 20°F peut être vu aussi bien comme une information que comme une donnée.

De la même manière, les capteurs ne mesurent que des forces brutes – il dépend du contexte d'interprétation de savoir s'il s'agit d'une pression, d'un poids, d'un ressort, etc. C'est uniquement au sein d'un modèle et d'une théorie construites

23. (Floridi 2010, 2011, 2017, 2019b).
24. Voir plus loin, § 9.
25. (Berry 2017, p. 71-72).

par les physiciens que ces mesures peuvent prendre des « valeurs », et que les calculs que réalisent les ordinateurs peuvent être spécifiés et leurs résultats dotés de signification.

L'information dont parle Berry est donc un élément de savoir qu'acquiert un agent interprétant une donnée – ou plutôt qu'il croit acquérir, puisque l'interprétation peut être incorrecte, et l'information fausse. Comme toutes les activités humaines requièrent du savoir – ne serait-ce que pour la lecture correcte de la situation dans laquelle agir – on comprend pourquoi l'informatique, selon Berry, la discipline occupée du « traitement de l'information », parvient à les pénétrer, voire à les transformer toutes.

On perçoit cependant qu'une telle thèse est fragile, car elle dépend du terme *interprétation* dont les philosophes connaissent l'ambiguïté. Avant de critiquer la thèse de Berry, il est utile d'en pointer d'abord le contenu positif, qui est visible assez directement dans les domaines où « l'information est numérique par construction ».

§ 6. Illustration de la thèse de Berry : Le cas de l'économie

L'exemple de l'économie est intéressant pour illustrer l'intuition de Berry. Prenons l'exemple d'un transfert d'argent (dans le cadre d'un achat, d'un salaire, d'un impôt, etc.), l'un des processus économiques les plus élémentaires. Lorsque l'agent économique A donne 100€ à l'agent B, cette opération de don ne signifie rien d'autre que le compte bancaire du premier doit être décrémenté de 100€ et celui du second augmenté d'autant. Le don d'argent peut être représenté par une double opération arithmétique *par construction*, car la monnaie a exactement cette signification, d'être une valeur pure d'échange (nous simplifions ici à outrance la théorie économique de la monnaie, mais cela suffit à notre propos). L'opération de transfert d'argent est elle-même construite par convention, ainsi que les institutions qui veillent à son enregistrement comme les banques, les intermédiaires de paiement, la banque centrale.

Il existe de nombreuses autres instances d'opérations qui peuvent ainsi être représentées par des informations construites spécialement à cet effet. L'une des inventions majeures de Facebook, le bouton « J'aime » en est un autre exemple. Il permet à un auditeur qui apprécie un billet (*post*) de véhiculer son sentiment. L'action de presser le bouton créée une information numérique très élémentaire, puisqu'elle ne véhicule aucune raison, ni aucune intensité de l'appréciation. Mais

cette simplicité a l'avantage de la rendre aisément manipulable, disponible pour d'autres actions. L'ensemble des « J'aime » reçus peuvent ainsi être comptabilisés, cette quantité peut être interprétée (« ce billet est populaire ») et susciter de nouvelles opérations, notamment des transactions publicitaires.

Dans l'informatique qui a trait aux échanges humains, qu'ils soient à teneur économique ou non, toute l'information est ainsi *construite*, comme le dit Berry, car les communications et les transactions, ainsi que leurs sous-jacents (biens symboliques ou matériels) sont eux-mêmes construits par l'activité humaine, sous l'égide d'institutions qui fixent la valeur des échanges. La puissance de l'informatique vient de ce qu'elle permet à ces valeurs d'être exprimées directement, de telle sorte que les opérations les concernant puissent s'exprimer selon leur logique propre. On parvient ici à la thèse centrale de Berry citée plus haut, que la puissance de l'informatique provient de ce qu'elle « rompt l'identification ancestrale entre type d'information et support physique ».

Ce principe est particulièrement pertinent dans le monde économique, et l'exemple du commerce électronique est ici parlant. Avant Internet, un magasin ou point de vente avait plusieurs fonctions : exposer les produits disponibles ainsi que leur prix, enregistrer des transactions (fonctions d'information), permettre au client de retirer son produit et de ce fait, stocker ces produits sur place (fonctions physiques). Du fait de la cherté des emplacements, un point de vente devait faire des compromis : proximité des clients, prix (fonctions de l'emplacement du magasin), profondeur du stock, largeur de l'offre (fonctions de la taille du magasin). L'arrivée d'Internet a permis de rompre ces compromis en proposant en ligne des offres très larges, alors que les stocks étaient déplacés vers de larges entrepôts situés dans des zones logistiques – loin des clients, mais avec des prix attractifs. Cette rupture est celle que mentionne Berry, entre la partie informationnelle et la partie physique des fonctions du point de vente. Comme le prédit justement un consultant de stratégie, Philip Evans, au début des années 2000 :

> Les flux d'information tendent à se séparer des flux physiques. […] La standardisation de la connectivité et de l'information entraîne des réductions progressives des coûts de transaction. Puisque les organisations [économiques] ont pour unique fonction d'économiser des coûts des transactions, [elles deviennent moins pertinentes] et les marchés se substituent donc à elles. […] La colle [qui tenait ensemble les chaînes de valeur] fond progressivement. Les édifices que sont la chaîne de valeur, la chaîne d'approvisionnement, les intermédiaires commerciaux, les organisations, etc. (que

> la stratégie conventionnelle d'entreprise prend tous pour des faits primaires) sont progressivement déconstruits[26].

Internet a ainsi entraîné des bouleversements profonds dans de nombreuses industries, car leur partie « informationnelle » a subi des reconfigurations importantes, souvent au profit de nouveaux acteurs. Ces bouleversements de déconstruction et de reconstruction peuvent être mis en perspective de l'analyse que Marx faisait de la substitution à l'ouvrier de la machine industrielle. Cette substitution, loin de se résumer à une augmentation brute de productivité à gestes équivalents, affranchit le processus de fabrication des contraintes corporelles humaines (vitesse, force, position des bras...) de telle sorte qu'il peut être reconceptualisé *à partir* de l'objet à fabriquer lui-même – et cela jusqu'à ce que l'atelier tout entier se trouve reconfiguré, ainsi que les relations économiques entre industriel, consommateur et ouvrier[27]. De la même manière, on peut avancer que l'informatique engendre une révolution économique car elle permet de reconfigurer en profondeur les chaînes de valeur informationnelles selon leur logique propre, en éliminant tous les compromis économiques qui avaient été historiquement réalisés du fait de leur dépendance à des substrats physiques.

Tout cela est bien connu, mais comment interpréter cette révolution ? La lecture que nous venons d'en donner, si on la généralise, peut suggérer une forme de dualisme. Toute activité humaine devrait être ainsi analysée en une partie informationnelle et une partie matérielle, non seulement les échanges économiques, mais également les techniques comme la poterie ou le tissage, les relations sociales, les rapports politiques, etc. D'une certaine manière, c'est bien à cette décomposition entre information et matière que procède l'informatisation de ces pratiques. Cela signifie-t-il pour autant que cette dualité préexiste à cette informatisation, que celle-ci ne ferait que révéler ? Devrait-on dire de toute pratique humaine qu'elle est un mélange d'information « immatérielle » et d'entités et de forces « matérielles » ? Que cela voudrait-il dire ? Mais si tel n'est pas le cas, *d'où* provient la possibilité d'informatiser ainsi les pratiques ?

Pour tenter d'y voir plus clair, il est utile prolonger l'examen de la thèse de Berry par la considération d'autres exemples.

26. (P. Evans 2000).
27. Voir (Marx [1867] 2006, p. 426) et le commentaire de (Séris 1994b, p. 159). Nous revenons sur cette idée à plusieurs reprises dans le cours de ce travail, notamment § 86.

§ 7. Le paradoxe de l'information

Dans le texte cité plus haut sur le codage de l'information, Berry ne nous indique que les différentes manières de *traduire* l'information en séries numériques. Il ne nous indique pas du tout ce qu'on peut faire avec ces séries. À en rester là, on pourrait penser que l'informatique sert uniquement à transmettre et à conserver les connaissances. Elle réalise bien sûr ces tâches, grâce à Internet et aux bases de données, mais elle fait aussi bien plus que cela.

Dit autrement, l'idée de coder l'information est courante depuis le télégraphe au moins, mais jusqu'à l'apparition de l'ordinateur, les codes ne servaient qu'à transmettre l'information, mais nullement à résoudre des problèmes variés grâce à ces codes. C'est ce que l'informaticien Peter Denning appelle le « paradoxe de l'information », et qu'il décrit ainsi :

> La théorie classique de l'information de Shannon démontre que l'information peut être transmise et reçue avec précision par des processus qui ne dépendent pas de la signification de l'information. Les ordinateurs étendent cette théorie, non seulement en transmettant mais en transformant l'information sans référence à la signification. Comment des machines qui fonctionnent indépendamment de la signification peuvent-elles générer de la signification pour les observateurs ? D'où viennent les nouvelles informations[28] ?

La réponse qu'apporte Denning à ce paradoxe est la suivante :

> Le processus de conception [du programme] transfère l'idée d'addition de nos têtes vers des schémas d'instruction qui effectuent l'addition. La signification de l'addition est conservée dans la conception de la machine et de ses algorithmes.
>
> Ceci vaut pour toute autre fonction calculable. Nous transférons notre idée de ce que signifie pour cette fonction de produire son résultat vers un programme qui contrôle la machine pour qu'elle réalise précisément cela. Nous transférons notre idée de la signification de la fonction vers la conception de la machine.
>
> De ce point de vue, l'idée que les machines et les systèmes de communication effectuent des traitements sans tenir compte de la signification des données binaires est fragile. Les algorithmes et les machines ont des significations implantées en eux par des ingénieurs et des programmeurs. Nous concevons des machines de sorte que la signification de chaque étape incrémentale, jusqu'au résultat, corresponde à notre intention, si les données entrées ont des significations également correspondantes[29].

28. (Denning et Bell 2012, p. 470).
29. (Denning, Martell et Cerf 2015, p. 50).

En d'autres termes, ce qui est bien plus fondamental que le codage de l'information, c'est la programmation d'opérations mécaniques qui correspondent aux opérations signifiées. Il semble bien que nous soyons revenus à notre point de départ. Nous cherchons d'où vient la possibilité de transférer dans des programmes les méthodes et les savoir-faire humains. Berry nous parle d'information et de codage, mais à l'analyse, cela ne s'avère être qu'une étape dans le cours de la programmation, et nullement l'explication de la manière dont elle transcrit les actions elles-mêmes dans les instructions données à la machine.

Cependant, si Berry passe sur cette explication, c'est peut-être que cette transcription est évidente ? Denning d'ailleurs ne semble pas penser que la programmation, telle qu'il vient de la définir, pose de problème conceptuel. La raison en est évidente : c'est qu'il n'y a là rien de très nouveau pour un scientifique, comme l'exemple de l'addition le laisse entendre : la représentation d'opérations concrètes dans un système formel où elles sont remplacées par un calcul « aveugle » est le procédé fondamental de l'algèbre, tel qu'il fut notamment conçu dans toute sa généralité par ceux qu'on appelle les algébristes anglais (Peacock, De Morgan, Boole) au début du XIX^{e} siècle[30].

Dans le cas de la science et de l'économie, l'algébrisation des pratiques humaines est déjà réalisée, et « les informations y sont numérisées par construction ». L'exemple des images et des sons, qui constituent l'essentiel de notre perception sensorielle, est immédiatement beaucoup plus difficile à interpréter. Berry nous explique comment les numériser, mais il ne nous dit pas comment « programmer » les opérations que nous faisons couramment avec ces objets. *De facto*, pendant très longtemps la numérisation des images et des sons s'est limitée à des applications extrêmement sommaires : transmission, compression, découpages et recollages, distorsions, analyses de fréquences, etc. Mais *ce que nous faisons* avec des images et des sons – reconnaître une personne, une situation, un paysage, ces opérations ne sont pas du tout aisées à programmer. Autrement dit, nous avons seulement affaire, avec la numérisation décrite par Berry, à des traitements de *données* : mais nous avons perdu en cours de route la capacité de les *interpréter* et de les constituer en informations véritables.

Ainsi, si on me montre deux photos, je saurai immédiatement dire, dans la majorité des cas, s'il s'agit de la même personne ou non. Une telle opération n'est devenue accessible aux ordinateurs que très récemment, les premiers systèmes com-

30. Voir à ce sujet l'article de (Durand-Richard 1990), notamment après la p. 145.

merciaux de reconnaissance faciale datant de la fin des années 1990. La raison en est que le codage numérique des images, tel que le décrit Berry, perd toute l'information « structurelle » que nous savons y reconnaître. Ce n'est même qu'avec l'apprentissage profond* – donc après 2010, que de telles opérations sont devenues fiables à grande échelle. On ne saurait d'ailleurs parler de *programmation* à leur propos, puisqu'on ne cherche plus ici à reconstituer la procédure (si procédure il y a) par laquelle nous reconnaissons une personne dans une image. On s'appuie sur un théorème mathématique, que toute fonction régulière d'un ensemble fini à un autre peut être approchée par des fonctions connues de manière de plus en plus fiable, à mesure qu'on accumule des observations à son propos. En supposant que l'acte par lequel nous reconnaissons des personnes dans des images peut être représenté par une telle fonction, on peut ainsi en développer des approximations algorithmiques.

Le cas des « forces mécaniques [qui] se numérisent grâce à des capteurs adaptés » est plus simple mais relève d'un principe encore différent. Mise à part leur étude scientifique, ces forces nous intéressent par le fait qu'elles assistent ou résistent à notre propre action physique dans le monde, que celle-ci passe directement par notre corps propre, ou par le biais d'outils ou de machines. Leur numérisation ouvre donc la voie à la programmation de cette action, qui permet à l'ordinateur de contrôler l'agent qui l'exécute, en général une machine.

Prenons un exemple concret, où la force en présence est tout simplement la température d'une pièce qu'on veut réguler. C'est le principe du thermostat, dont une version simplifiée peut être représentée par le programme suivant, qui utilise la syntaxe du langage Python :

Code 1.1 – Code Python d'une fonction thermostat

```
1  while True:
2  a = temperature()
3  if a > cible + tolerance : commande (froid)
4  elif a < cible - tolerance : commande (chaud)
5  else: commande(arret)
6  time.sleep (regulation)
```

Dans ce code[31], `cible` désigne la température cible, exprimée en degrés,

31. Par convention dans tout notre travail, tous les termes coloriés sont internes au langage de programmation considéré ou à ses librairies*, tandis que ceux en noir sont définis par le programmeur (même si cette définition n'apparaît pas dans le fragment de code considéré).

`tolerance` l'amplitude autorisée autour de cette cible, exprimée en une valeur absolue de degrés, et `regulation` une durée, exprimée en secondes, pendant laquelle le thermostat doit attendre avant de lancer une nouvelle mesure (ligne 6). La fonction `temperature()` (ligne 2) fait appel à un capteur de température, qui lui renvoie une valeur que la procédure enregistre dans la variable `a`. Le mots clés `if, elif` et `else`, aux lignes 3, 4 et 5, testent cette valeur selon les trois cas de figure possibles afin de donner l'intruction adéquate, indiquée après la ponctuation « : » la fonction `commande()` peut actionner le climatiseur selon trois valeurs possibles `chaud, froid, arret` (lignes 3 - 5) : chauffer, refroidir ou s'arrêter respectivement. Cette procédure est itérée indéfiniment, ce qui est indiqué par la condition toujours vérifiée `while True`, en ligne 1.

Ce programme est effectif – il permet de réguler la température de la pièce – d'abord parce qu'il est relié au monde externe par les fonctions `temperature()` et `commande()`, et ensuite parce qu'il existe une relation physique entre l'action du climatiseur et la température. Cette relation est fondée sur des processus physiques qui se déroulent dans le monde, dont nous avons une connaissance pragmatique ou scientifique. On doit noter que l'efficacité du programme – la vitesse à laquelle il amène la pièce à la bonne température – dépend du bon calibrage de ses variables, notamment son temps d'attente `regulation`, qui doit dépendre de la puissance du climatiseur et de l'écart de température constaté. L'algorithme présenté pourrait être très largement optimisé pour assurer la convergence de la température et éviter son oscillation entre le chaud et le froid. D'une manière générale, l'effectivité du programme dépend principalement de sa modélisation correcte de phénomènes physiques.

§ 8. De l'information à la connaissance

Les trois exemples examinés frappent donc par la diversité des principes par lesquels les ordinateurs s'insèrent dans nos pratiques humaines. D'un côté, cela n'est pas étonnant : les exemples traitaient d'activités différentes – économique, perceptive, physique – qui ne se situent absolument pas sur le même plan. C'est parce que les activités économiques sont conventionnelles et construites qu'elles se programment. C'est parce qu'il y a des principes physiques à l'œuvre dans le monde que nous pouvons confier le contrôle de nos actions matérielles à des ordinateurs. C'est parce qu'il y a une régularité sous-jacente à notre perception discriminante de notre environnement qu'on peut l'approcher par des fonctions mathématiques.

La difficulté de la programmation est par ailleurs croissante dans cette série. Il y a également des activités humaines que nous ne savons pas encore programmer ou traiter statistiquement comme, quoiqu'on en dise, la conduite de voiture ou la conception de grands projets. Il n'y a pas de raison non plus de penser que les principes qui viennent d'être d'énoncés sont les seuls à l'œuvre dans l'informatisation de notre monde humain.

Le principe proposé par Berry – l'informatique est puissante car toute information est codable numériquement – semble donc insuffisant ; le codage de l'information est une condition nécessaire, mais nullement suffisante de l'informatisation de nos pratiques, dont les causes sous-jacentes semblent être diverses et de complexités différentes. Ainsi le célèbre informaticien Donald Knuth critique le terme *informatique* en Français et *Informatik* en Allemand, en ce que « ces noms mettent en valeur la « matière » que les algorithmes manipulent (l'information ou les données) plutôt que les algorithmes eux-mêmes[32] ».

Ce qu'indique cependant cette condition, c'est que la programmation a affaire de manière centrale avec la connaissance. Si les programmes, qui sont les produits de la programmation, manipulent des informations et en génèrent de nouvelles, alors la programmation elle-même se rapporte à celles-ci comme au second degré, puisqu'elle s'occupe d'agencer ces processus mêmes de manipulation et de création d'informations, c'est-à-dire – toujours selon Berry – les processus mêmes de la connaissance.

L'hypothèse à laquelle on est amené est donc la suivante : c'est parce que nos activités doivent accéder à des savoirs positifs sur le monde ou, plus généralement, sur le problème à résoudre, que l'informatique parvient si bien à les assister. La programmation transposerait sur des machines nos processus de connaissance – calcul, mémoire, perception, comparaison, etc. – et c'est pour cela qu'elle aurait la polyvalence – voire l'universalité – de la connaissance elle-même. C'est cette hypothèse que nous examinons à la section suivante.

1.3 Informatique et connaissance

Dans cette section, nous étudions deux interprétations majeures de la relation entre programmation et connaissance. D'abord, la science cognitive à ses débuts fit l'hypothèse que le fonctionnement de l'esprit devait être conçu comme

32. (Knuth 1974, p. 324).

une exécution de programmes inscrits, d'une manière ou d'une autre, dans la matière cérébrale. Admettre cette « hypothèse computationnelle de l'esprit » semble rendre notre questionnement aussitôt caduc. En effet, si la compétence cognitive de la personne humaine est semblable à celle d'un ordinateur, alors il n'y a pas à s'étonner qu'on puisse, de manière plus ou moins aisée, transposer des pratiques humaines dans des programmes informatiques : cette possibilité découle en droit de leur nature même, d'être des formes de programmes. C'est d'ailleurs exactement cela qu'Herbert Simon chercha à démontrer dans ses expériences de psychologie cognitive, que les processus de pensée humains peuvent être modélisés par des programmes informatiques[33]. Au-delà des difficultés propre à cette thèse, nous tentons de montrer qu'elle ne répondrait que de manière superficielle à notre questionnement (§ 9).

Plus prudent, le mouvement pédagogique de la *pensée computationnelle* (*computational thinking*) affirme seulement que la programmation, au sens large, est un modèle pour la pensée rationnelle. Son enseignement serait donc bénéfique à la formation de l'esprit en général, et aiderait la réflexion dans des domaines et des contextes qui lui sont totalement étrangers. Cependant cette perspective peut être interprétée de manière caricaturale, et elle ne s'appuie pas sur une théorie de la connaissance à proprement parler (§ 10). Dans le développement suivant, nous dégageons notre propre hypothèse, que la programmation est une forme de connaissance, ainsi que notre problématique générale (§ 11).

Une remarque terminologique est ici nécessaire. Nous allons dans ce travail utiliser de temps à autre, et à contre-cœur, l'adjectif *computationnel*. Outre que ce terme n'est pas encore attesté dans le dictionnaire – *calculatoire* serait sa traduction exacte – il souffre d'une ambiguïté fâcheuse : il peut désigner aussi bien les calculs numériques, qui se réduisent toujours à des calculs sur les entiers naturels (et donc à des manipulations de valeurs binaires), que toute procédure pouvant être exécutée par un ordinateur (*computer*). Dans notre perspective, on le verra, ces deux sens doivent être soigneusement distingués. Nous gardons cependant le terme *computationnel* en l'état, car il est devenu standard, même en français, pour désigner certaines notions ou théories (*hypothèse computationnelle de l'esprit, pensée computationnelle, science computationnelle, pan-computationnalisme, modèle computationnel*), etc. Il est entendu que nous ne l'utilisons qu'en référence à celles-ci, et non pour l'incorporer au lexique de notre travail.

33. Voir par exemple les études de cas résumées dans (Simon et Newell 1972).

§ 9. L'hypothèse computationnelle de l'esprit

Cette idée – que la programmation semble pouvoir répliquer la complexité de tous les comportements, parce qu'elle porte sur la connaissance elle-même – suscita l'enthousiasme d'une nouvelle génération de psychologues, au début des années 1950, et donna naissance à la science cognitive, dans sa première version, dite *computationnelle*. La psychologie américaine, à cette époque, était dominée par le béhaviorisme, qui avait forgé le concept de *comportement* comme objet d'étude propre de la psychologie, le distinguant aussi bien de la physiologie du système nerveux que de l'étude des états mentaux. Le béhaviorisme visait à expliquer le comportement par des processus de fixation de réactions sensori-motrices (« stimulus - réponse ») face à des situations récurrentes et par leur composition subséquente en réactions complexes. Les béhavioristes en étaient cependant venus à admettre l'idée que l'intelligence, même animale, ne saurait s'expliquer par la simple plasticité de ces mécanismes internes, et qu'une certaine capacité d'autoreconfiguration devait leur être conférée. Ainsi Edward Tolman, un psychologue béhavioriste, propose dans les années 1940 la notion de carte cognitive (*cognitive map*) dont la métaphore, qui prépare celle de l'ordinateur, est celle du standard de commutation téléphonique. Un standard téléphonique n'est pas seulement le lieu central où passent tous les chemins possibles, il est surtout le lieu où *tel* chemin est *sélectionné* plutôt qu'un autre[34].

L'idée de programme fut à ce titre une libération conceptuelle pour les jeunes psychologues comme George Miller et Jerome Bruner. Plutôt que se contenter de l'image étriquée d'une composition mécanique de circuits comportementaux entrée-sortie, la cybernétique et l'informatique montraient la puissance de circuits à rétroaction pour modéliser à peu près tout ce qu'on voulait. Comportements simples et complexes, compositions de comportements, convergence de comportements aléatoires vers un comportement stable, la mémoire, le langage, etc. – l'informatique montrait la possibilité de tout cela avec une clarté confondante, sans rien sacrifier au parti-pris mécaniste imposé par le béhaviorisme à la psychologie, garant de sa scientificité. L'idée de *programme* permettait même de réhabiliter les notions d'*intention*, puisqu'un programme qui en appelle un autre peut apparaître comme la finalité de ce dernier, et de *réflexivité*, puisqu'un programme peut en analyser et en modifier un autre.

34. (Tolman 1948).

Il s'agissait donc d'inverser la perspective : plutôt que de partir de l'étude de rats et de pigeons pour comprendre comment des comportements élémentaires pouvaient progressivement se composer en comportements complexes, en prenant pour paradigme l'image du circuit élémentaire entrée - sortie, il fallait partir tout de suite du comportement complexe dans toute sa généralité, capable de s'auto-reconfigurer dans une certaine mesure – le comportement cognitif humain, en prenant pour paradigme l'image du programme informatique, qui contenait d'emblée la possibilité de programmes plus simples modélisant l'instinct, l'habitude, la compétence, etc. La psychologie devenait ainsi *essentiellement* cognitive, au sens où, par ce terme, il ne s'agissait plus seulement de désigner cette branche de la psychologie s'occupant des phénomènes de cognition, mais d'expliquer tout type de comportement à partir de la cognition.

Ces idées furent exprimées dans toute leur clarté – et avec une certaine naïveté – par Miller, Galanter et Pribram ([1960] 2013), dans un ouvrage fondateur, au titre explicite : *Les Plans et la structure du comportement* (*Plans and the Structure of Behavior*). Ils y affirment que le fonctionnement de l'esprit repose sur l'exécution de *Plans* qui sont, « pour un organisme, essentiellement la même chose qu'un programme pour un ordinateur[35] ». C'est même toute l'architecture de l'ordinateur moderne qui est convoquée pour expliquer l'intention et la décision humaines, comme on peut le percevoir à travers ce court extrait :

> Quelque chose d'important se passe lorsqu'une décision est prise d'exécuter [un Plan]. Il est tiré de la mémoire morte (*dead storage*) et placé en contrôle d'un segment de notre capacité de traitement de l'information. Il est placé dans la visée de l'attention et, tandis que nous commençons à l'exécuter, nous engageons également une série de tâches mineures mais nécessaires, comme rassembler l'information, nous rappeler la progression dans la réalisation du plan à chaque instant, etc.[36]

Nous avons résumé ailleurs[37] les vicissitudes historiques et les difficultés intellectuelles de cette thèse, déjà maintes fois exposées par d'autres auteurs. Nous nous contentons ici de rappeler trois difficultés majeures qui ont émergé du long débat sur l'hypothèse computationnelle.

La première difficulté est de déterminer comment l'être humain se donne des problèmes et des objectifs, comment il choisit lesquels sont importants ou urgents à résoudre, et lesquels ne le sont pas. Les solutions darwiniennes (tous les pro-

35. (Miller, Galanter et Pribram [1960] 2013, p. 16).
36. (ibid., p. 64).
37. (Stephanou 2022).

blèmes se ramènent à assurer sa survie et celle de l'espèce) se heurtent à l'objection qu'un esprit computationnel pourrait choisir la stratégie de la paramécie pour accomplir ces fins, c'est-à-dire se contenter d'une niche écologique très avantageuse mais n'exhibant aucune des propriétés intéressantes de la rationalité. Il faut donc imaginer un système élaboré de besoins ou de valeurs, qui inclut notamment des paramètres sociaux, et qui devrait donc être au moins partiellement évolutif. Les difficultés de cette idée émergent de manière assez évidente de la discussion récente et très optimiste sur le sujet, entre Le Cun et Dehaene, experts respectivement en intelligence artificielle et en science cognitive[38].

La seconde difficulté est le *problème du cadrage* (*frame problem*)[39]. Il est formulé initialement comme un problème technique par les chercheurs en intelligence artificielle, mais ses présupposés philosophiques sont progressivement explicités et montrés comme problématiques. Il devient le cheval de bataille de Dreyfus ([1972] 1992) dans sa critique de l'intelligence artificielle. Ce problème concerne la construction d'un plan d'action quelconque. Lorsqu'on cherche à présenter cette construction comme une déduction à partir d'axiomes logiques, on s'aperçoit qu'il ne suffit pas d'expliciter les changements positifs qu'une action provoque dans l'environnement ; il faut également expliciter ce qu'elle ne change pas – or le nombre d'axiomes nécessaires à cela croît de manière linéaire avec les propriétés de l'environnement. Ainsi, *« ouvrir le robinet »* n'implique pas seulement que *« l'eau coule »*, mais également que *« l'évier reste à sa place »*, que *« les murs de la cuisine restent de la même couleur »*, etc. La solution qui semble évidente, de proposer que *« rien n'est modifié par une action si cela n'est pas explicitement spécifié »*, conduit à des paradoxes. Par exemple, si on a empilé des cubes les uns sur les autres, et qu'une action fait tomber le cube situé à la base de la pile par terre, le système ne révisera pas automatiquement la série de propositions précédemment établies, que chaque cube est situé au-dessus de l'autre. De nombreux systèmes logiques furent élaborés pour trouver une solution générale de ce problème, sans succès probant jusqu'à aujourd'hui. Une seconde approche a consisté, dans les années 1980, à construire une base de données des « faits du sens commun », qui pourrait servir d'axiomatisation à tout problème de ce type, mais cette entreprise titanesque montra seulement qu'il fallait disposer d'une ontologie complète du

38. Voir (Dehaene, Le Cun et Girardon [2018] 2020), notamment chapitres 6 et 8.

39. Nous nous appuyons ici principalement sur la bonne synthèse de (Gryz 2013), sauf concernant son interprétation de la position de Dreyfus. Voir également (F. M. Brown 1987) pour un aperçu des recherches logiques sur le sujet à la fin des années 1980.

monde afin que les règles de déduction associés à ces axiomes puissent également être définis. Les philosophes, et notamment Dreyfus, reformulèrent finalement le problème comme une question de *pertinence* : comment sélectionne-t-on, lorsqu'on agit, les faits pertinents pour notre prise de décision ? Pour Dreyfus, cela ne peut être réalisé que par un agent *qui a déjà un monde*, c'est-à-dire qui construit activement son interprétation de la situation, y compris de ce qu'il doit et peut faire.

La troisième difficulté est le *problème du fondement symbolique* (*symbol grounding problem*)[40]. Il peut être vu comme une extension du problème précédent[41], bien qu'il soit formulé de manière différente, puisqu'il prend son origine dans la critique adressée par Searle (1980) à l'intelligence artificielle. Un système formel, quel qu'il soit, ne peut espérer reproduire les compétences de l'intelligence car il n'a aucun rapport avec le monde qu'il est censé représenter ou avec lequel il doit interagir. S'orienter intelligemment dans le monde, pour un tel système, serait comme vouloir apprendre le chinois à partir d'un seul dictionnaire monolingue : on pourrait établir de nombreuses observations et règles concernant les rapports des idéogrammes entre eux, mais rien ne permettrait de relier ces règles au monde auquel la langue chinoise se réfère. Les systèmes de symboles doivent être nativement couplés à une interaction de l'agent avec le monde, et refléter les règles de cette interaction.

Rapportées à notre première question directrice, ces questions se résument en une seule : Que serait un programme qui aurait une capacité universelle à écrire des programmes ? Programmer c'est, comme on l'a dit, bien plus que « coder » une spécification dans un langage formel. C'est écrire cette spécification elle-même et c'est ensuite s'assurer que le monde réel (l'environnement du programme) est prêt à interagir avec lui. On voit mal comment un système formel qui ne connaît rien du monde aurait la capacité de se fixer lui-même des problèmes à résoudre, de les cadrer correctement, et enfin de s'assurer que sa solution formelle se rapporte bien, *en ce moment*, à ce monde qu'il ne connaît pas. Derrière ces trois problèmes apparaît donc en creux la figure du programmeur, qui *sait* tout cela.

L'hypothèse computationnelle de l'esprit connaît une certaine survivance aujourd'hui, dans de nombreuses variantes qui contournent plus ou moins ces difficultés. En particulier, il est aujourd'hui généralement admis dans ces théories que

40. (Harnad 1990).
41. (Harnad 1993).

les algorithmes statistiques adaptatifs, comme les réseaux de neurones, jouent un rôle essentiel dans l'émergence de la connaissance, ce qui marque une forme de retour au béhaviorisme. L'hypothèse computationnelle est par exemple explicitement soutenue par la philosophie de l'information de Luciano Floridi, qui qualifie les personnes humaines d' « inforgs » (organismes informationnels) essentiellement similaire aux ordinateurs qui l'« ont rejoint dans l'infosphère[42]. »

Le présent travail ne prend pas position pour ou contre ce type des thèses, car *en droit* cela n'est pas requis par son propos. Nous cherchons en effet seulement à expliquer ce qui rend la programmation effective dans la résolution de nombreux problèmes que se posent les personnes humaines. Cette question est relative à d'*autres* manières de résoudre des problèmes qui se passent d'ordinateurs. Que toutes ces compétences, la programmation et les autres facultés cognitives humaines, puissent ultimement être toutes caractérisées comme des calculs, des mécanismes, des phénomènes émergents, etc. – pour autant que ces assertions aient un sens – ne nous aidera nullement à distinguer ce qu'il y a de spécifique à la programmation elle-même *relativement* à ces autres facultés. Elle ne nous permettra pas, par exemple, de comprendre pourquoi il est plus aisé de programmer une pratique scientifique qu'une pratique sociale, et quelles sont les difficultés concrètes que la programmation rencontre dans ces projets. Nous devons partir de la « chose même », non des principes généraux dont elle serait supposée dépendre.

L'hypothèse computationnelle de l'esprit ne peut donc guider notre questionnement. Elle nous intéresse cependant pour une seconde raison, plus positive : c'est qu'elle affirme le lien profond entre connaissance et programmation, et ce faisant, promeut une vision intrigante de la connaissance, entièrement *opérationnelle*, construite progressivement à partir de l'idée de *contrôle du comportement* d'un organisme dans un environnement – très éloignée des conceptions de la philosophie classique de la connaissance comme correspondance de la pensée avec le réel – et qu'elle partage, comme on va le voir, avec d'importants courants philosophiques du début du xx^{e} siècle.

L'observation centrale de la psychologie computationnelle cognitive nous semble donc être celle-ci, que de nombreuses pratiques rationnelles des personnes prennent *souvent* la forme de procédures, qui présentent une ressemblance indéniable avec des programmes informatiques. Dans le cadre de la résolution *réfléchie* de problèmes, et malgré les biais cognitifs qui peuvent les affliger, les

42. (Floridi 2019a, p. 210).

personnes tentent, dans certaines circonstances qui requièrent par exemple prudence, consensus ou réitération fréquente, d'élaborer puis de suivre des procédures déterminées, fiables, adaptables aux circonstances, et communicables aux autres personnes. Ces procédures peuvent être apprises et se transformer en habitudes.

La piste ouverte par ces réflexions nous semble bonne : il semble y avoir une parenté entre l'établissement de ces pratiques rationnelles et la programmation, entre les procédures de la vie courante et les procédures informatiques. Quelle est cette parenté ? *A minima*, il nous semble nécessaire d'inverser la perspective des premiers cognitivistes : l'élaboration de plans et de procédures n'est certainement pas une forme élémentaire de programmation, mais au contraire, c'est cette dernière qu'il est peut-être possible de décrire celle-ci comme une forme très spéciale, voire extrême, de planification. Cette hypothèse suffirait à expliquer pourquoi elle peut, dans bien des cas, répliquer les plans et les procédures de la vie courante selon ses propres modalités, et se montrer ainsi effective. Il y aurait ainsi à l'œuvre, dans tous ces cas de figure, une même forme de réflexion qui porterait sur la *logique des actions et des interactions*, et que nous pouvons appeler de manière provisoire *raison procédurale*.

§ 10. La pensée computationnelle

Ce n'est pas seulement dans le cadre de réflexions psychologiques et philosophiques que le lien entre programmation et connaissance a été relevé. Il fut également affirmé avec force par les informaticiens eux-mêmes, qui éprouvèrent comme une surprise la rigueur requise par la programmation, bien qu'ils fussent déjà rompus à la planification et à l'organisation de très longs calculs par des calculateurs humains, et aux erreurs typiques qui peuvent y survenir. L'historien de l'informatique Priestley décrit ainsi l'expérience du premier bogue :

> Lorsque le célèbre mathématicien et calculateur expert Douglas Hartree configura l'ENIAC en 1946 pour résoudre un système d'équations différentielles, la réponse de la machine à une situation inattendue le surprit. Le calcul impliquait une variable r, qui était connue d'après ses propriétés physiques pour être toujours positive et qui était stockée sous la forme d'un entier compris dans la plage 0–99. Cependant, à un moment du calcul, r avait pris de manière inattendue la valeur –1. L'ENIAC interpréta cela comme son complément, 99, et continua à calculer avec cette valeur erronée. Hartree, s'exprimant quelques semaines seulement après l'événement, en

> était visiblement assez ébranlé, disant : « Je vois que j'aurais dû prévoir la possibilité que cette situation se produise [...] Je ne l'avais pas prévue malgré trente ans d'expérience en calculs de toutes sortes. »[43]

Il est important de noter ici que ce n'est pas la longueur du calcul qui le rend difficile à spécifier correctement, mais le type d'erreurs que peut faire la machine, qui diffèrent de celles qu'on peut attendre d'un opérateur humain. Aussi, Hartree insistera régulièrement, par la suite, sur la nécessité de prendre le « point de la vue de la machine » en programmant.

Cela est exprimé de manière quelque peu mélancolique par un grand informaticien anglais, Maurice Wilkes, se rappelant ses premières expériences de programmeur en 1949 :

> Dès que nous avons commencé à programmer, nous avons découvert à notre grande surprise qu'il n'était pas aussi facile que nous le pensions d'obtenir des programmes corrects. [...] Je me souviens de l'instant exact où j'ai réalisé qu'une grande partie de ma vie serait désormais consacrée à trouver les erreurs dans mes propres programmes[44].

Il y a donc ici un paradoxe, entre l'évidence que présente un programme une fois écrit, et la difficulté que nous avons à le concevoir. Le caractère surprenant de cette difficulté est qu'elle n'est pas due à des facteurs externes, comme pourrait l'être celle d'un chemin de montagne, d'une expérience de physique ou d'une négociation commerciale. Le calcul semble être justement une activité qui aplanit toutes les difficultés, puisqu'il ne s'agit que d'y manipuler des symboles inertes selon des règles parfaitement bien définies. Or cette compétence de planifier dans leur moindre détail des calculs, suggère Priestley, est d'une fausse simplicité pour l'esprit humain : elle lui est au contraire éminemment artificielle, et elle ne s'acquiert qu'au terme de nombreuses années d'apprentissage.

A contrario, la réussite, après une longue suite d'échecs, peut faire naître chez le programmeur passionné une certaine jubilation, qu'exprime parfaitement l'informaticien Frederick Brooks :

> [Il y a] la fascination de fabriquer des objets énigmatiques complexes, constitués de pièces mobiles finement imbriquées les unes aux autres, et de les voir fonctionner en cycles subtils, déroulant les conséquences de principes qu'on y avait intégrés dès le commencement. [...] Il y a [également] le plaisir de travailler une matière aussi maniable. Le programmeur, comme le poète, ne travaille qu'à une distance minime

43. (Priestley 2021, p. 93).
44. (Wilkes 1979).

> de la pure pensée. Il construit ses châteaux dans les airs [...] Pourtant, le programme construit, contrairement aux mots du poète, est réel dans le sens qu'il se meut et qu'il travaille, produisant des résultats visibles distincts de la construction elle-même[45].

L'expérience de cette surprise, du niveau insoupçonné de rigueur dans le raisonnement qu'exigeait la programmation, ainsi que le sentiment d'accomplissement qu'entraîne la réussite d'un programme, a ainsi amené nombre d'informaticiens à émettre la thèse que programmer était une manière privilégiée de connaître. Alan Perlis énonça très tôt cet aphorisme célèbre :

> Vous croyez savoir quand vous apprenez, vous en êtes plus sûr quand vous savez rédiger, encore plus quand vous pouvez enseigner, mais certain seulement quand vous pouvez programmer.
> *You think you know when you learn, are more sure when you can write, even more when you can teach, but certain when you can program*[46].

La remarque de Perlis portait sans doute sur des méthodes de résolution de problèmes mathématiques, puisqu'elle est citée la première fois par le mathématicien Forsythe (1959, p. 656) dans le cadre d'une discussion sur les mérites qu'aurait un cours de programmation pour des étudiants en mathématiques appliquées.

Dans cet article, Forsythe avait décrit les mathématiques comme étant des « outils mentaux polyvalents » (*general-purpose mental tools*). Donald Knuth, commentant Forsythe, étend à l'informatique cette caractérisation et revient à partir de là à l'aphorisme de Perlis, qu'il développe de la manière suivante :

> On a souvent dit qu'une personne ne comprend pas vraiment quelque chose jusqu'à ce qu'elle l'apprenne à quelqu'un d'autre. En fait, une personne ne comprend pas vraiment quelque chose tant qu'elle ne peut pas l'enseigner à un ordinateur, c'est-à-dire l'exprimer sous forme d'algorithme. « L'ordinateur automatique force vraiment cette précision de pensée qui est censée être le produit de toute étude des mathématiques » (Forsythe p. 655). La tentative de formaliser les choses sous forme d'algorithmes conduit à une compréhension beaucoup plus profonde que [la] manière traditionnelle. Les linguistes pensaient qu'ils comprenaient les langues, jusqu'à ce qu'ils essaient d'expliquer les langues aux ordinateurs ; ils ont vite découvert combien il leur restait encore à apprendre. Beaucoup de gens ont mis en place des modèles informatiques de choses, et ont découvert qu'ils apprenaient plus en mettant en place le modèle qu'en regardant réellement la sortie du programme final[47].

45. (Brooks 1982, p. 7).
46. (Perlis 1982).
47. (Knuth 1974, p. 326).

Hartmanis (1981, p. 1), un autre théoricien renommé de l'informatique, reprend cette idée de Knuth :

> Je vois l'informatique comme une nouvelle discipline scientifique qui construit un ensemble complètement nouveau de conceptualisations, de théories et d'applications, [...] un arsenal intellectuel dont les concepts et les modèles intellectuels peuvent être empruntés par les informaticiens et les scientifiques d'autres disciplines pour réfléchir à leur problèmes spécifiques et développer leurs propres théories.

En 1981, lors d'une table ronde prospective sur l'avenir de l'informatique, incluant d'ailleurs Hartmanis, un autre informaticien, Reddy, propose une formulation différente :

> L'informatique est une science consacrée à l'amélioration des compétences mentales de l'être humain[48].

Le concept central ici est celui d'*amélioration*, sur lequel Reddy insiste en rapprochant l'informatique de deux autres disciplines ayant la même vocation méliorative : d'une part l'ingénierie, qui se consacrerait à l'extension des compétences *physiques* des êtres humains, et d'autre part la médecine, qui s'occuperait de « réparer la machine humaine ».

Cette définition vaste de l'informatique comme transformant les « techniques cognitives » des êtres humains est devenue courante depuis. Elle anticipe en effet la vague de la *pensée computationnelle*[49] qui se répand en sciences de l'éducation depuis une quinzaine d'années. Ce courant pédagogique avance que la programmation développe les compétences cognitives générales de l'individu, car elle oblige à structurer la réflexion, à penser précisément, à réfléchir de manière critique à ses erreurs. Elle doit être enseignée dans les troncs communs d'enseignement, non pas tant pour former des informaticiens, que pour améliorer la capacité générale des étudiants à résoudre des problèmes. Jeannette Wing, qui a lancé ce courant, insiste que ces concepts, comme la modularité, le parallélisme*, la randomisation*, la redondance*, la sûreté*, etc. sont des outils pertinents pour décrire le monde, aussi bien dans la vie professionnelle que dans la vie de tous les jours.

Wing intègre également la science computationnelle (sur laquelle nous revenons plus loin) à ce vaste mouvement :

> Nous sommes témoins de l'influence de la pensée computationnelle sur d'autres disciplines. Par exemple, l'apprentissage automatique a transformé les statistiques. [...]

48. (Traub 1981, p. 354).
49. (Wing 2006). Pour un aperçu historique, (Tedre et Denning 2016).

> La contribution de l'informatique à la biologie va au-delà de sa capacité à explorer de vastes bases de séquences [d'ADN]. L'espoir est que les structures de données et les algorithmes – nos abstractions et nos méthodes de calcul – puissent représenter la structure des protéines de manières nouvelles afin d'élucider leur fonction. La biologie computationnelle change la façon de penser des biologistes. De même, la théorie informatique des jeux change la façon de penser des économistes ; la nano-informatique, celle des chimistes ; et l'informatique quantique, celle des physiciens[50].

Quel crédit faut-il donner à ces prétentions de l'informatique de constituer un nouveau modèle pour la connaissance, tant dans ses méthodes que dans ses contenus ? Ce modèle doit-il être compris comme s'étendant à toute forme de connaissance rationnelle, ou seulement à une partie ? Et, dans ce cas, comment délimiter celle-ci ? Par ailleurs, s'agit-il d'une simple reformulation d'une conception traditionnelle de la connaissance ou d'une révolution épistémologique qui est ici proposée ?

Une interprétation trop directe de ce caractère exemplaire de la programmation pour la pensée rigoureuse peut évoquer l'hypothèse computationnelle de l'esprit. Par exemple Wing n'hésite pas à décrire nos habitudes cognitives les plus élémentaires *par* des concepts informatiques, comme dans ce passage :

> Considérez ces exemples du quotidien : Lorsque votre fille va à l'école le matin, elle met dans son sac à dos les choses dont elle a besoin pour la journée ; c'est du pré-chargement et de la mise en cache*. Lorsque votre fils perd ses mitaines, vous lui proposez de revenir sur ses pas ; c'est un retour sur trace* (*backtracking*). À quel moment arrêtez-vous de louer des skis et achetez-vous une paire ? ce sont des algorithmes incrémentaux* (*online algorithms*). Dans quelle file d'attente vous placez-vous au supermarché ? c'est de la modélisation des performances pour les systèmes multi-serveurs[51].

On ne peut manquer de remarquer le caractère artificiel, voire pédant, de ces exemples. Même s'il n'est pas affirmé ici que l'esprit humain « calcule », c'est toute la description de ses processus de réflexion qui est normée par une nomenclature informatique, qui énoncerait leur vérité sous-jacente. Par là est suggérée une pétition de principe similaire à celle de l'hypothèse computationnelle de l'esprit : comme la pensée rigoureuse est intrinsèquement programmatrice, il n'y a pas à s'étonner que la programmation soit effective à modéliser et répliquer toutes nos

50. (Wing 2006, p. 34).
51. (ibid., p. 34).

pratiques, puisque celles-ci n'en sont que des approximations.

Une autre interprétation, que nous tenterons d'étayer au cours de ce travail, est cependant possible. Loin d'être un modèle idéal de la pensée auquel se conformeraient plus ou moins bien nos esprits ou nos cerveaux imparfaits, les concepts informatiques seraient le *résultat d'un travail de clarification* de certaines de nos manières de penser, des méthodes que la réflexion a progressivement épurées à partir de l'observation des pratiques intellectuelles.

Nous revenons sur cette hypothèse plus loin dans cette introduction. À ce stade, nous voulons seulement retenir des assertions des informaticiens et des pédagogues évoqués que la programmation développe une certaine manière radicale de connaître, qui a trait à la résolution effective de problèmes, de quelque nature qu'ils soient. C'est cette hypothèse que nous devons explorer de manière critique dans ce travail.

§ 11. Problématique

Parvenus à ces questions, il est utile de récapituler le chemin parcouru jusqu'ici.

Nous sommes partis d'un questionnement concernant la présence généralisée des ordinateurs dans notre monde, qui semble tenir à la polyvalence de leurs usages. Contrairement aux autres grandes inventions de la révolution industrielle, les ordinateurs n'ont pas une fonction précise, mais semblent tout simplement effectifs à résoudre de nombreux types de problèmes auxquels sont confrontés les personnes dans leurs activités. Ayant rejeté la réponse qu'« il y a besoin de calcul dans toutes les activités humaines », nous avons émis une première hypothèse :

1. L'ordinateur est une machine polyvalente parce qu'elle est programmable.

La programmation, à ce stade, désigne simplement la transcription des méthodes ou procédures connues, servant à résoudre des problèmes, en instructions que saurait suivre un ordinateur. Mais une telle assertion pose immédiatement d'autres questions : existe-t-il un langage universel pour décrire *tout type de procédures* ? Et comment des procédures, qui sont de pures significations, pourraient-elles être « comprises » par des machines ?

L'idée qu'il est possible de coder numériquement tout type d'information permet de clarifier partiellement le problème, mais elle n'explique pas l'essentiel, c'est-à-dire la capacité de coder les activités mêmes qui manipulent ces informations.

L'inspiration que fut la notion de *programme* pour les sciences cognitives ainsi

que les assertions répétées des informaticiens, que leur activité concerne essentiellement la connaissance, nous conduisent à une seconde hypothèse :

2. La programmation est une activité qui radicalise une forme de connaissance rationnelle.

Cette connaissance rationnelle a affaire à la résolution de problème, dans le sens très large que nous avons vu : il s'agit de résoudre, de manière très large, des situations d'insatisfactions qu'on sait décrire de manière plus ou moins précise. Une telle formulation semble rapprocher la programmation de ce qu'on appelle dans le langage courant la connaissance pratique. Cependant, une telle association n'est pas évidente, puisque la programmation sert également les mathématiques et les sciences.

Il convient de noter ici que nous ne nions pas que la programmation peut servir à autre chose qu'à la résolution de problème : les jeux électroniques et l'art numérique en sont des exemples. Cependant, ces usages de la programmation sont hors du champ de notre questionnement, qui s'interroge sur l'effectivité des ordinateurs à *intervenir* dans notre monde humain ; par ailleurs, dans les jeux et l'art, il n'est pas question de connaissance, au moins au sens où nous l'entendons ici.

Les deux hypothèses élaborées se démarquent de nombreuses idées couramment invoquées pour expliquer les succès de l'informatique, ainsi :

1. Les ordinateurs sont des machines à calculer, et le raisonnement est une forme de calcul.
2. Les ordinateurs sont des machines à traiter l'information, et toute connaissance est une forme de traitement de l'information.

Pour résumer les arguments développés ci-dessus, nous nous accordons avec ces idées en ce qu'elles mettent le raisonnement et la connaissance au centre de l'informatique. Cependant, ce n'est pas l'*exécution* elle-même des processus informatiques qu'il nous semble falloir rapprocher de la connaissance, mais bien leur *programmation*. Par ailleurs, nous évitons les pétitions de principe, qui assignent d'emblée une nature profonde au raisonnement et à la connaissance. Nous ne cherchons pas à savoir si toute connaissance est un programme, mais quelle forme *spécifique* de connaissance est *programmer*. À partir de là seulement, se dégageront peut-être des enseignements plus généraux concernant la connaissance dans son ensemble.

Une troisième hypothèse doit être ajoutée aux deux premières afin de boucler notre questionnement :

3. La forme de connaissance qu'est la programmation doit expliquer comment et en quel sens des machines peuvent être effectives, *c'est-à-dire contribuer à la résolution de situations* humaines *problématiques.*

En d'autres termes, cette troisième hypothèse signifie que ce n'est pas en regardant dans le détail la configuration technique des machines que nous comprendrons par quel miracle elles peuvent raisonner avec nous, voire sans nous, mais en examinant *ce qu'est* cette forme de raisonnement que nous leur prêtons.

C'est à l'exploration et à la précision de ces trois hypothèses que ce travail est consacré. Parmi ces trois hypothèses, la seconde a naturellement un rôle prééminent dans notre recherche, qui justifie le titre que nous lui avons donné : « *La programmation est-elle une forme de connaissance ?* »

1.4 La connaissance des procédures

Dans cette section, nous explicitons notre question principale. Nous montrons d'abord la difficulté qu'il y a à *classer* l'informatique comme discipline de savoir (§ 12) et suggérons que c'est parce qu'elle a trait à la connaissance des procédures. Dans les deux développements suivants, nous montrons le caractère problématique d'une telle assertion, d'abord sur le plan historique : comment se fait-il qu'on commence à étudier ces objets de connaissance si tardivement ? (§ 13), puis sur le plan conceptuel (§ 14).

§ 12. L'informatique entre théorie et pratique

La question que nous venons de formuler semble supposer qu'on a à disposition un catalogue de formes de connaissance auxquelles on pourrait comparer la programmation. De tels catalogues sont certes fréquents dans l'histoire de la philosophie. À l'époque moderne, ils prennent souvent la forme d'arbres, comme celui de l'*Encyclopédie* de Diderot et d'Alembert[52], inspiré lui-même de celui de Bacon. Cependant, il n'est malheureusement pas aisé d'y insérer une nouvelle discipline. Par exemple, dans ce catalogue, les techniques sont rangées parmi l'histoire naturelle, la logique parmi les sciences de l'homme (l'art de penser), et les mathématiques parmi les sciences naturelles. L'informatique, participant de ces trois disciplines, se retrouverait ainsi écartelée dans ce tableau.

52. Volume I, *Discours Préliminaire*, p. XLVIII.

Cette difficulté devint manifeste lorsque, dans les années 1950-1960, on chercha à situer l'informatique dans le champ disciplinaire des cursus universitaires.

Notamment, l'idée qui peut sembler naturelle aujourd'hui, de la placer parmi les disciplines techniques, fut l'objet d'intenses débats jusqu'à la fin des années 1990, dont Tedre (2014) rend compte. Un tel positionnement incitait en effet à modeler l'enseignement et la recherche informatiques sur les cursus d'ingénierie. Cette orientation était encouragée par les praticiens de l'industrie, confrontés à l'explosion des coûts et des problèmes de qualité liés au développement de logiciels complexes, mais elle ne pouvait convenir aux théoriciens, qui souhaitaient maintenir l'ancrage de leur discipline fermement dans les mathématiques[53]. Un troisième courant, pragmatique, proposait de placer l'informatique à mi-chemin, dans le domaine des sciences empiriques, en se prévalant de sa littéralité (en langue anglaise : *computer « science »*), et en arguant que toute science avait une double dimension, théorique et pratique. Il fallait cependant expliquer qu'une science puisse porter sur des artefacts, ce que tenta de faire Simon ([1969] 1996c) dans les *Sciences de l'Artificiel*, en rapprochant l'informatique de sciences comme l'économie, le droit ou la médecine.

Ces disputes se sont depuis apaisées. Un premier consensus possible est d'affirmer que l'informatique participe de ces trois divisions du savoir. Primiero (2020) par exemple, les utilise pour organiser le plan de son manuel d'introduction aux fondements de l'informatique. Par ailleurs, l'approche interdisciplinaire des *STEM* (*Science, Technology, Engineering and Mathematics*)[54] qui devient centrale depuis vingt ans dans les cursus académiques, rend ces disputes moins pertinentes, d'autant plus que le « T » de *Technology* est en pratique identifiée à l'informatique, signifiant bien son rôle distinct et central dans cette interdisciplinarité.

Malgré ces consensus pragmatiques récents, ces débats révèlent des tensions internes à l'informatique elle-même. La programmation est une *activité informatique* qui produit des programmes et des systèmes informatiques. Il semble bien alors que son savoir propre doive être appelé *savoir technique* ou *pratique*, par opposition au savoir mathématique que produit l'informatique dite *théorique*, habituellement considérée comme une branche des mathématiques.

Le problème de fond est que l'opposition entre théorie et pratique, par laquelle

53. Nous développons cette perspective au cours de ce travail, section 11.1.
54. Voir (Mohr-Schroeder, Cavalcanti et Blyman 2015, p. 3-14) pour un court panorama introductif.

on décrit traditionnellement les deux pans complémentaires d'une science[55], ne fonctionne pas ici très bien. En particulier, le rapport habituel de prééminence de la théorie sur la pratique ne tient pas. Dans la vision courante, la théorie établit des lois ou des théorèmes sur des phénomènes dont la réalité est indépendante des interventions des praticiens. Ceux-ci peuvent utiliser ces lois ou ces théorèmes pour concevoir leurs méthodes de travail, mais les théoriciens ne s'y intéressent en général qu'occasionnellement, lorsqu'ils font de la « science appliquée » ou lorsqu'une méthode pratique révèle un phénomène nouveau et digne d'intérêt. Même dans le cas des sciences où les interventions pratiques motivent étroitement les élaborations théoriques, comme l'économie, la médecine, ou la chimie, les objets fondamentaux de la théorie (la production et l'échange de richesses, le corps humain, les liaisons moléculaires) en restent évidemment distincts. La situation est complètement différente en informatique, car les savoirs des informaticiens théoriques ne portent sur rien d'autre que les méthodes des praticiens (les algorithmes) dont ils établissent des propriétés, ou alors sur des méthodes de second degré, comme les méthodes formelles, qui permettent aux praticiens de mieux travailler. Aussi est-ce ce point précis que Tedre (2014) identifie comme le point nodal des disputes disciplinaires concernant l'informatique :

> [Dans l'histoire de l'informatique], le point de débat le plus récurrent est celui entre théorie et pratique, qui se matérialisa […] selon des dimensions diverses comme science et art, académique et industriel, scientifique et technique, général et particulier, pur et appliqué, et bien d'autres[56].

On pourrait objecter que les classements des savoirs ont toujours une certaine dose d'arbitraire, et qu'ils reflètent bien plutôt des orientations culturelles ou idéologiques que l'essence de la connaissance. Cependant, dans le cas de la programmation, la difficulté de son classement parmi les connaissances n'est pas superficiel, et reflète la multiplicité des manières par lesquelles on peut caractériser son objet. Nous en présentons trois :

À un premier niveau, on peut dire que la connaissance du programmeur porte sur les algorithmes. Par exemple, si on connaît l'algorithme du tri à bulles*, l'un des plus simples (et des moins efficaces) pour ordonner une séquence d'éléments sur lesquels existe une relation d'ordre, comme des nombres ou des mots, on peut être dit avoir une connaissance effective concernant ce problème, puisqu'en l'ap-

55. À ce sujet, voir plus loin, § 111.
56. (Tedre 2014, p. 208).

pliquant on le résout de manière satisfaisante.

À un second niveau, il peut s'agir de savoir programmer cet algorithme sur un ordinateur particulier, doté d'un langage particulier. Ici, entrent en jeu des connaissances concrètes, concernant le fonctionnement de cet ordinateur, la syntaxe du langage, etc. Par exemple, certains ordinateurs ou langages imposent des bornes à la taille des nombres et des mots : il faut en tenir compte dans un algorithme de tri.

À un troisième niveau, on peut exiger du programmeur non seulement qu'il sache implémenter des algorithmes dans des environnements informatiques donnés, mais également qu'il identifie, en amont de cela, quel algorithme choisir en fonction du problème à résoudre. Si par exemple il s'agit de proposer à un utilisateur trois chemins possibles pour effectuer un voyage, un algorithme de tri sera certes nécessaire pour hiérarchiser tous les chemins envisageables selon des critères de préférence. Mais là ne se situe pas l'enjeu réel de la programmation, qui doit plutôt déterminer quels critères pertinents considérer (distance, vitesse, coût), comment les pondérer, ou s'il faut engager un dialogue avec l'utilisateur afin qu'il précise ses préférences. Dans ce dernier cas, se pose la question de *l'effectivité du dialogue lui-même*. Il faut que l'utilisateur et le programme se comprennent mutuellement afin que le problème soit résolu effectivement, et cela suppose de nouvelles connaissances, comme celle des habitudes de communication de l'utilisateur. On voit que la programmation implique la connaissance d'autre chose que des algorithmes et des ordinateurs : ici, celle d'une certaine portion du monde (ce qu'est un voyage, un chemin, de ses critères d'évaluation), et celle des habitudes de communication des utilisateurs. Dans des exemples plus complexes, comme la programmation d'un logiciel de gestion pouvant être utilisé par des milliers de personnes au sein d'une organisation, et intégrant des centaines de fonctions différentes à travers plusieurs domaines (finance, achats, production, ventes, etc.), une connaissance fine de l'organisation tout entière est ainsi souvent requise du programmeur qui, dans ces cas de figure, n'est bien sûr pas un individu isolé mais une équipe informatique parfois constituée de dizaines de collaborateurs.

De plus, ces trois niveaux par lesquels on peut définir l'objet de la programmation ne sont pas hermétiques l'un à l'autre, et on peut voir comment passer continûment de l'un à l'autre. Par exemple un algorithme de tri peut être pris abstraitement, comme nous l'avons fait, ou dans la particularité du classement des mots d'une langue spécifique. On peut le programmer dans un langage adossé

à une machine virtuelle*, comme Java, qui le rend plus ou moins indépendant du système informatique sur lequel il sera exécuté. On peut ensuite ajouter à ce programme des spécifications formelles qui traduisent les conditions concrètes de son utilisation, de manière à arriver progressivement à la connaissance riche du problème à traiter. Ainsi, ce qui ressemblait initialement à l'opposition entre informatique théorique et pratique semble pouvoir se résorber à un simple gradient d'abstraction concernant le problème à résoudre.

Il semble donc y avoir dans la programmation une singularité qui remet en cause notre opposition traditionnelle entre connaissance théorique et pratique et qui la réduit à une simple différence d'abstraction.

Cette singularité semble due au fait que la connaissance propre de la programmation a pour objet une certaine forme de méthodes – *celles qui permettent de résoudre des problèmes*, dans l'acception très générale que nous donnons à ce terme[57]. Concernant la connaissance de ces objets singuliers, l'opposition entre théorie et pratique semble inopérante. En tant qu'elle guide l'action, une telle méthode peut parfois n'être rien d'autre que la simple formalisation d'un savoir-faire, d'une compétence pratique. Mais lorsqu'elle prend la forme précise et générale d'un algorithme, elle semble être un objet mathématique, dont l'effectivité peut être prouvée rigoureusement, par des moyens purement théoriques.

Dans ce travail, nous appellerons *procédures* de telles méthodes de résolution de problèmes. Ce choix, s'il est conforme à l'usage actuel de ce terme, n'est pas évident. Le lien entre les notions de *procédure* et *méthode* est complexe, comme nous allons le montrer au prochain développement par un premier survol de leur évolution. Il faut noter également qu'ici, nous entendons *procédure* en un sens bien plus large que celui qu'il a en informatique d'une *procédure effective*, notion que nous analysons dans le cours de ce travail[58]. Une procédure a le même degré de généralité que le problème qu'elle résout, et peut être moins précise que lui. Lorsqu'on vous indique un chemin en vous conseillant « *d'aller jusqu'au carrefour doté d'un panneau lumineux, de prendre la deuxième à droite, puis de continuer jusqu'à la zone industrielle, et de tourner juste avant sur la première ruelle à gauche* », vous disposez bien là d'une procédure, mais il n'est pas du tout évident que vous réussissiez à l'appliquer de manière *effective*. Plus généralement encore, nous ne limitons pas la *procédure* à la forme impérative (une suite d'« instructions ») qu'elle

57. Voir plus haut, § 3.
58. Voir plus loin, section 11.1.

prend souvent. Une fonction mathématique qui donne le résultat d'un problème est dans notre sens une procédure ; il en est de même pour un ensemble de clauses logiques. La légitimité de tels rapprochements s'éclairera au cours de ce travail.

Les difficultés à classer l'informatique parmi les sciences et les techniques, entre la théorie et la pratique, ne font donc que révéler un problème bien plus large, qui est celui de ce que pourrait signifier *connaître des procédures*, un sujet qui a peu attiré l'attention des philosophes.

§ 13. Procédures et méthode

Il peut sembler paradoxal que la connaissance des procédures n'ait pas été l'objet d'une réflexion significative en philosophie de la connaissance, tant elle semble commune. Une piste possible d'explication de cela est que ce savoir lui-même n'a jamais été perçu comme problématique *en tant que tel*. Il s'inscrit dans notre expérience la plus élémentaire du monde humain, qui requiert que nous sachions agir et réagir selon l'instinct ou l'habitude face à certaines situations typiques. En tant qu'une méthode ne fait qu'expliciter *a posteriori* ce savoir-faire, à des fins d'enseignement et de transmission, elle ne semble donc pas être non plus un objet problématique. Ainsi le *Système des Connaissances* de l'*Encyclopédie*, évoqué plus haut, classe les techniques parmi les connaissances d'histoire, c'est-à-dire comme faisant appel à la mémoire plutôt qu'à la raison.

Ce n'est que depuis le début du XX^{e} siècle que la dimension théorique de ces objets a été explicitement dégagée – d'abord par le programme formaliste de Hilbert, qui constitua les preuves comme objets théoriques à part entière, puis par les articles de Church et Turing dans les années 1930 qui dégagèrent plus généralement la notion théorique de *procédure effective*, et permit l'éclosion d'une nouvelle discipline mathématique, la théorie de la calculabilité. Ces « découvertes » mettent en avant le contenu *rationnel* qu'il y a dans la connaissance d'une procédure, par opposition à sa dimension simplement mnémotechnique.

C'est de ce fait qu'il faut s'étonner : pourquoi faut-il attendre si longtemps pour que ce contenu rationnel des procédures soit reconnu ? Si cette connaissance est inscrite dans une forme de raison procédurale qui dépasse largement le cadre de l'informatique, alors celle-ci aurait dû être aperçue et thématisée par les philosophes ou les logiciens, d'une manière ou d'une autre, avant l'arrivée des ordinateurs. Donald Knuth exprime ainsi cet étonnement :

L'informatique aurait dû exister bien avant l'apparition des ordinateurs[59].

Si en effet l'informatique est (selon lui) l'étude des algorithmes, qui sont eux-mêmes simplement « des séquences de règles disant comment produire un résultat donné à partir de données initiales, en un nombre d'état fini », pourquoi ne se constitue-t-elle comme science qu'au moment de l'avènement historiquement contingent de machines capables de les exécuter automatiquement ?

Knuth s'empresse de corriger son étonnement en citant les algorithmes babyloniens de la plus haute antiquité. Cependant là n'est pas le problème. On peut en effet trouver la trace de procédures dès l'apparition de l'écriture, dans tous les domaines du savoir. Jim Ritter (2010) montre que les textes babyloniens décrivant les *pratiques rationnelles* mathématiques, juridiques et médicales partagent une même structure grammaticale sous-jacente, notamment par un emploi standardisé de prépositions et de formes verbales qui permet de restituer la logique des procédures, telle que nous la connaissons aujourd'hui, qui implique la mémorisation de données, de résultats intermédiaires, de branchements, voire d'opérations parallèles.

Ce qui est étonnant, étant donné cette réalité, est que le concept *transverse* de procédure n'ait pas pour autant émergé pour désigner d'une manière commune ces objets élémentaires du savoir, et que les termes désignant ces prescriptions soient restés, jusqu'au début du XX^e^ siècle, spécifiques à chaque discipline (*algorithme, construction, remède, recette* , etc.). Il n'est pas plausible de répondre que cela nécessitait une capacité d'abstraction qui n'était pas à la portée des Anciens.

Ici, on pourrait contrer que ce concept existe bien, dès l'Antiquité grecque au moins, où il est véhiculé par le terme transverse de *méthode*. Il est donc utile d'anticiper nos études futures de la troisième partie afin de montrer la non-évidence d'une telle assimilation.

Pour Aristote, *μέθοδος* signifie d'abord *investigation* et *démarche d'investigation*[60]. Nous montrerons, dans notre chapitre consacré au Stagirite, qu'il y a bien chez lui un début de réflexion procédurale, notamment dans les *Topiques* et les *Premiers analytiques*, mais elle reste cloisonnée à la logique et n'a pas de postérité au-delà du Moyen Âge.

Le terme *μέθοδος* n'est aussi que très rarement utilisé (non plus que ses variantes *ἔφοδος*, *ὁδός*) par les mathématiciens grecs. Seuls Ptolémée et Héron, au

59. (Knuth 1974, p. 323).
60. Voir (Joachim Ritter et al. 1980, p. 1306) et plus loin dans ce travail, la section 6.3.

début de notre ère, l'emploient pour désigner des heuristiques mais également certains algorithmes numériques. Une explication possible de cette rareté serait le dédain général que les mathématiciens grecs avaient pour la *logistique*, c'est-à-dire la technique du calcul, qu'ils distinguaient fermement de l'arithmétique, science des nombres. Les procédures numériques, destinées à produire des résultats, auraient ainsi été associées au monde de l'artisanat et du commerce que les « hommes libres » dédaignaient[61]. À ce dédain est également associé une certaine conception de la vérité, qui associait les procédures pratiques au domaine du contingent et donc de l'incertain. Ainsi, Ptolémée, dans son introduction à l'*Almageste*, affirme que les mathématiques seules sont capables de certitude – par opposition non seulement aux disciplines pratiques et techniques, mais également à la physique et à la théologie, autres disciplines théoriques – et cela du seul fait de la rigueur de leurs méthodes[62]. Il ne serait pas étonnant que, dans ce cadre de pensée, il n'ait pas été possible de percevoir la parenté profonde qui existe entre des procédures de calcul ou de démonstration et des prescriptions pratiques ayant cours dans les techniques ou la médecine.

À l'époque romaine, le terme *méthode* connaît une certaine fortune en médecine, puisqu'il dénomme une école spécifique, l'école méthodique, opposée aux dogmatiques et aux empiristes. Le terme reste ainsi courant dans cette discipline, et au Moyen Âge il est d'une manière générale associé à l'art bien plus qu'à la théorie de la connaissance[63]. Dans ce contexte, se référant à son étymologie (ὁδός = route, voie), les philosophes scolastiques l'associent à la notion d'efficacité, en la définissant comme un « raccourci » parmi les routes possibles pour faire quelque chose. Cette idée de *route* ou de *chemin* qui guide l'action « pas à pas », de manière déterminée, sûre et efficace, peut certainement être vue comme une préfiguration de notre procédure moderne.

Cependant, lorsque La Ramée réintroduit le concept de *méthode* en logique, il l'éloigne de ce sens technique et procédural. Pour lui, la méthode traite uniquement de l'ordre dans lequel on doit exposer une science *déjà constituée*, notamment en vue de fins pédagogiques. Elle n'a rien à faire avec l'invention, c'est-à-dire avec l'investigation des causes. La Ramée, dans une image célèbre, la montre à l'œuvre dans la reconstitution de l'ordre des propositions de la grammaire, qu'on aurait

61. (Asper 2009).
62. Ptolémée, *Syntaxica Mathematica (Almagest)*, H5, trad. Toomer, p. 36.
63. Voir (Hamou 2014, p. 2), (Joachim Ritter et al. 1980, p. 1305-1308).

écrites sur des petites bandes de papier, et mélangées dans une urne. Le problème qu'il s'agit de résoudre ne concerne ni la recherche de ces propositions (l'invention), ni le contrôle de leur validité (le jugement), puisqu'elles sont déjà données. La méthode consiste uniquement à les ordonner, en partant des principes les plus généraux jusqu'aux propositions les plus particulières. Cet emploi très spécifique du terme l'éloigne de l'idée technico-pratique de *procédure*, puisqu'il n'y a, ici, nulle *production* de savoir, ou de progrès de données à des résultats – il y a au mieux, une méta-procédure qui transforme la présentation du savoir.

C'est seulement progressivement, au début du XVII^e siècle, que l'idée de méthode est mobilisée pour désigner à nouveau les règles qui doivent conduire l'investigation. Hamou (2014) avance que Descartes joue un rôle décisif dans cette inflexion, en lui donnant un tour particulier. En quelque sorte, la méthode cartésienne concerne bien l'apprentissage et la pédagogie, comme le voulait La Ramée. Cependant il ne s'agit pas ici d'apprendre un corpus inerte de propositions déjà formées, mais, pour un esprit, de se diriger lui-même dans sa recherche de la vérité, ce qui ne peut se faire qu'en adoptant l'ordre imposé par la clarté et la distinction des idées elles-mêmes. Descartes suit bien l'inspiration de La Ramée en ce sens, mais il conçoit l'esprit comme son propre Maître, qui doit s'enseigner lui-même. La méthode cartésienne n'est donc pas un ensemble de règles méta-théoriques, une logique de l'invention ou de l'exposition de la connaissance, mais d'abord « une pratique qui vise à aiguiser les facultés de l'esprit, à le rendre pour ainsi dire plus conscient, plus attentif[64] ». Cette inflexion éloigne encore plus l'idée de *méthode* de celle de procédure pratique.

Il y a cependant une seconde ligne de réflexion concernant la méthode, issue également de La Ramée, et qui présente plus d'affinité avec notre sujet. Bacon, ici, peut en être le point de repère. Son *Novum Organum* se propose de concevoir un « nouvel outillage » de la pensée qui, contrairement à ce que croit Descartes, lui est nécessaire afin de la dégager de ses propres illusions et préventions. Cet outillage comprend des instruments, mais aussi des tables de données, des diagrammes, des encyclopédies, toutes ces inventions graphiques permises notamment par l'imprimerie. Comme le dit Giglioni :

> Une méthode pour Bacon n'est pas de nature intellectuelle, elle ne doit pas non plus être une création de l'intellect, mais quelque chose qui lui fournit des aides externes. En ce sens, c'est une *machina*, un appareil capable de restaurer notre perception

64. (Hamou 2014, p. 6).

> claire de la vraie nature des choses. […] La technologie est le remède de l'*ingenium*. Un boiteux sur une route sûre arrivera plus facilement à destination qu'un athlète courant dans un territoire sauvage ; en utilisant des compas et des règles, une personne ordinaire tracera des cercles et des lignes droites avec plus de précision qu'un artiste bien formé à la main seule ; […] Il ne peut y avoir de véritable changement ou réforme (*instauratio*) que si le pouvoir d'innovation (c'est-à-dire l'*ingenium*) et la capacité de l'esprit à élaborer des modèles de représentation de la réalité (c'est-à-dire l'*intellectus*) sont canalisés et exploités par des moyens extra-mentaux contrôlant la manière dont nous recueillons des informations et gardons le contrôle de nos passions (*appetitus*)[65].

Une telle vision de la méthode comme d'un outillage externe à l'esprit, voire comme d'une machine, prépare celle de Leibniz, qui s'approche sans doute le plus de notre idée de *procédure* lorsqu'il exige que la méthode guide l'esprit à la manière d'un calcul. Il est significatif qu'il ne trouve cependant pas de terme pour exprimer cette idée et qu'il doive recourir à une métaphore :

> La véritable méthode nous doit fournir un *filum Ariadnes*, c'est-à-dire un certain moyen sensible et grossier, qui conduise l'esprit comme sont les lignes tracées en géométrie, et la forme des opérations qu'on prescrit aux apprentifs (sic) en Arithmétique. Sans cela notre esprit ne saurait faire un long chemin sans s'égarer. Nous le voyons clairement dans l'Analyse, et si nous avions des caractères tels que je les conçois en métaphysique et en morale, et ce qui en dépend, nous pourrions faire en ces matières des propositions très assurées et très importantes ; nous pourrions mettre les avantages et désavantages en ligne de compte lors qu'il s'agit d'une délibération[66].

La méthode dont veut se démarquer Leibniz dans ce passage est celle de Descartes, qu'il jugeait vide[67]. Mais, en gardant le singulier (« la véritable méthode ») auquel celui-ci tenait, Leibniz s'oriente vers l'idée de Caractéristique Universelle qui, d'une certaine manière, tourne le dos à notre concept moderne de programmation. Celle-ci vise en effet à déterminer une procédure adaptée pour chaque problème, qui pose lui-même les termes dans lesquels il doit être résolu. Dans le projet grandiose de Leibniz au contraire, un seul calcul universel, une seule procédure, doit permettre de résoudre toutes les questions, du moment que celles-ci seraient analysées en

65. (Giglioni 2022).
66. *Lettre à Jean Gallois*, sept 1677, in *Sämtliche Schriften und Briefe* (Darmstadt), vol. II.1 : Philosophische Briefwechsel (par la suite abrégé en D.II.1), p. 381.
67. *Lettre à Swenligius*, in *Philosophische Schriften* (éd. Gerhardt), vol. IV (par la suite abrégé en G.IV), p. 329.

leurs significations primitives et absolues, directement opératoires.

Reste néanmoins ici l'idée, commune à La Ramée, Bacon et Leibniz, que la méthode et ses outils peuvent *dégager* l'esprit du besoin d'une constante attention dans son investigation, en fournissant des soutiens à la mémoire, en abrégeant certains enchaînements, en le guidant vers les idées correctes. Tout cela pointe vers l'idée d'une mécanisation possible de certaines étapes de la réflexion, une idée qui trouve, chez La Ramée et Leibniz, son modèle dans celle de *calcul*.

Cependant, le Grand Siècle reste très éloigné de notre concept contemporain de *procédure*. Ce concept couvre tout type d'activité, tandis que la Méthode ne porte que sur les entreprises de connaissance. Elle est d'ailleurs conçue comme unique, tandis que nos procédures sont spécifiques à chaque problème. Elle vise à guider l'esprit à forger des connaissances, et donc à former sa propre autonomie de pensée, alors que les procédures ne traitent que des données, et ne font que mettre l'esprit en vacance. Ces différences majeures empêchent de voir dans les réflexions des Temps modernes sur la Méthode l'anticipation de cette raison procédurale que nous cherchons.

L'idée d'instruire une suite absolument rigoureuse d'opérations parfaitement définies reste ainsi confinée aux domaines de la logique et des mathématiques. Au chapitre 5, nous argumenterons que c'est de la réflexion économique que viendra le chaînon manquant permettant d'accéder à la conception générale de *procédure*. L'idée de la division du travail, popularisée par Adam Smith, encourage à décomposer les pratiques productives humaines en gestes élémentaires standardisés. Cette idée sera développée et précisée par Beckmann, Babbage, Marx dans leurs réflexions sur la technologie – jusqu'à ce que l'idée de procédure normée devienne centrale dans la conception moderne du travail avec Frederick Taylor. C'est par cette voie, où Babbage joue un rôle central, que la convergence conceptuelle aussi bien que concrète du calcul et de la procédure est finalement atteinte, dans les vastes bureaux organisant la production normée et intensive de calculs pour le compte des scientifiques, des militaires, des administrations et des entreprises dans le courant du XIX^{e} siècle[68].

Quant à la notion jumelle de *programme*, prise dans son sens informatique, elle précède également de peu l'avènement de l'ordinateur. Comme *procédure*, son glissement de sens se prépare dans le cadre de la révolution industrielle. Le terme est employé à partir des années 1920 pour qualifier la technologie qui sert à la

68. Nous revenons sur ces points au chapitre 5.

transmission de programmes de radiophonie (d'où le glissement de sens), et surtout, dès le XIX^e siècle, pour désigner des alarmes prédéfinies régissant les horaires des usines, les chemins de fer, etc. dont l'utilité générique motive le développement de « machines à programme » configurables, et capables de séquencer les opérations de machines industrielles[69]. L'entreprise de diriger les activités humaines par des séquences minutieusement réglées d'opérations bien définies date donc, globalement, du début du XX^e siècle.

L'esquisse que nous venons de tenter des évolutions sémantiques de ces trois termes, *procédure, méthode, programme* est certainement parcellaire. Elle suffit cependant à montrer combien il est peu évident de postuler que la programmation s'inscrit dans la continuité naturelle d'une « raison procédurale » organisant nos savoir-faire quotidiens et techniques. S'il a bien toujours existé des procédures mathématiques, juridiques, médicales, etc. il semble qu'une rupture conceptuelle ait été nécessaire afin qu'un *retour réflexif* sur ces pratiques devienne possible, qui permette d'en dégager le concept générique. Cette rupture précède de peu l'apparition de l'informatique, et elle est liée aussi bien à l'avènement des méthodes industrielles de production qu'à des progrès décisifs en méta-mathématique. Autrement dit, la programmation semble requérir un arrière-plan intellectuel et technique très particulier, qui la distingue radicalement des pratiques procédurales « de premier degré » attestées depuis les débuts de l'écriture.

§ 14. Deux questions directrices

Cette question historique est donc aussi une question conceptuelle : quelle est la nature de cette rupture qui semble faire apparaître un nouveau champ de connaissance, ou peut-être une nouvelle modalité de connaître ? Et en quelle mesure remet-elle en cause notre conception classique de la connaissance ? Deux questions peuvent être posées concernant l'idée d'une *connaissance de procédures* :

1. Quel rapport entretient cette connaissance de *ce qu'il faut faire* pour résoudre tel ou tel problème avec la connaissance simple de *ce qui est* ? En est-elle dérivée (une « application »), ou au contraire est-elle un mode autonome de connaissance, qui se constitue selon des principes propres ? Si oui, lesquels sont-ils, et quel serait son critère de vérité ou de validité ?
2. Comment des procédures pourraient-elle augmenter notre capacité de connaître elle-même – ainsi que le suggère l'importance que prend

69. (De Mol et Bullynck 2021, p. 36).

l'informatique en sciences et en mathématiques ? Cela signifie-t-il que la connaissance a le caractère d'un *faire*, d'une pratique comme une autre ?

Nous appellerons ces deux questions *directrices* ; elles vont en effet guider notre réflexion épistémologique et sont donc instrumentales à notre hypothèse principale concernant la programmation comme forme de connaissance.

Comme on le voit, ces questions s'ancrent directement dans l'ancien problème du rapport entre théorie et pratique. En d'autres termes, non seulement les méthodes ou procédures sont des objets de connaissance pour lesquels cette distinction classique est inopérante, comme nous l'avons noté plus haut, mais l'élucidation de leur statut épistémique semble étroitement liée à celle de la nature même de ce rapport. Notre problématique doit donc ici s'élargir dans cette direction. Nous commençons par présenter d'une manière très générale ce problème classique de la théorie de la connaissance, avant d'examiner quelques-unes de ses expressions contemporaines à la section suivante.

Le problème du rapport entre théorie et pratique consiste à évaluer si cette distinction est essentielle, c'est-à-dire si elle concerne deux formes radicalement distinctes de connaissance, ou si elle est seulement pragmatique ou culturelle – utile par exemple pour regrouper approximativement ce qu'on pourrait appeler, en empruntant ce terme à Wittgenstein, différentes « familles » de connaissance, comme la famille des sciences et celle des techniques, sans cependant que ce regroupement pointe vers des différences épistémiques nettes et déterminées.

Aristote avait affirmé avec force la première solution dans l'*Éthique à Nicomaque* (VI.1), allant jusqu'à distinguer deux âmes rationnelles, l'une occupée de la considération des êtres nécessaires, et l'autre de la délibération concernant les choses contingentes. Cette distinction fut immédiatement problématique, notamment à propos du savoir technique, sur le statut duquel le Stagirite lui-même semble hésiter : la technique est d'une part ancrée dans les savoir-faire pratiques, mais d'autre part possède une dimension réflexive qui lui permet de raisonner sur les causes de sa propre efficacité, comme le font les sciences à propos des phénomènes naturels. Aussi Kant, dans la célèbre introduction à la *Critique de la faculté de juger*, affirma-t-il contre Aristote l'identité de principe de la connaissance *rationnelle* pratique et théorique, traitant les préceptes de la technique et de la prudence de simples « corollaires » de la connaissance par concepts naturels[70].

70. Nous revenons longuement sur les différents sens de *technique* et *pratique* en section 4.4. Il est néanmoins important de préciser immédiatement que, dans ce travail, nous utilisons *pratique*

La primauté de principe de la science sur la technique fut renforcé au XIX^e siècle par l'émergence du discours technologique, où le développement des techniques en milieu industriel fut sommé de tirer parti des résultats des sciences et d'appliquer la méthode scientifique [71].

Cette position ne naît pas cependant avec les Lumières et la Révolution industrielle : elle s'ancre dans une ancienne tradition rationaliste de la connaissance. Si la connaissance vise à atteindre le vrai ou le réel à l'aide de la raison, elle requiert que le sujet se dégage de ses rapports contingents à l'objet considéré – particularité du point de vue, préconceptions, préjugés, intérêts : le savoir se construit en opposition à ces déterminations du sujet. Mais dans ce cas, ne faut-il pas dire que la connaissance pratique, qui est essentiellement intéressée, ne peut être *connaissance* que de manière dérivée, c'est-à-dire seulement en tant qu'elle se trouve fondée, ultimement, dans l'appréhension rationnelle du monde ?

Cette vision classique a cependant été régulièrement contestée par des philosophes qui, prenant pour point de départ les pratiques humaines et les interactions de l'être humain avec son milieu, se demandaient au contraire comment et pourquoi la connaissance pouvait en émerger, affirmant ainsi la primauté du savoir pratique et technique, qui naît de l'expérience corporelle du geste réussi, de l'attention portée à la configuration concrète des choses à manipuler et de l'intelligence des situations. De ce point de vue c'est, bien plus que le savoir-faire pratique, la connaissance théorique qui doit être expliquée, c'est-à-dire sa possibilité, son utilité, et les raisons de la forme très particulière qu'elle prend.

Notre hypothèse, que la programmation est une forme de connaissance, ne peut ignorer ces problématiques, sous peine d'être traitée de manière parcellaire ou partiale, en s'appuyant sur des thèses contestables, comme par exemple : *« la programmation est une activité théorique, puisque les mathématiques sont des activités théoriques »*, ou au contraire : *« la programmation est un savoir-faire, puisque*

en son sens originaire le plus large – ayant trait à l'action, et non dans le sens plus limité qui a cours parfois en philosophie, suite à Kant qui entendait le réserver à la philosophie morale. Kant utilise *technique* pour l'emploi le plus fréquent que nous aurons de *pratique*, qui vise d'une manière générale l'action ordonnée à la réalisation d'une fin.

71. Schatzberg retrace l'émergence de la *technologie*, tout à la fois discours idéologique et transformation des pratiques d'innovation technique au XIX^e siècle. L'ampleur de la contribution réelle des sciences au développement industriel est une question très débattue par les historiens des techniques, dans laquelle nous n'entrons pas (voir par exemple (FORMAN 2007)) pour une discussion polémique à ce sujet.

toute technique s'ancre dans un rapport pragmatique au problème à résoudre ». Tous ces termes – *théorie, pratique, technique, savoir-faire* – doivent donc être considérés dans ce travail comme eux-mêmes problématiques, et ne peuvent aucunement être des points d'appui pour notre réflexion. Au contraire, nous faisons l'hypothèse que la clarification de la nature du savoir qui a lieu dans la programmation peut en retour éclairer ces problèmes d'une lumière nouvelle. Comme nous l'avons remarqué plus haut, la possibilité d'étudier mathématiquement les procédures n'a été découverte qu'il y a un siècle, et les conséquences de ce fait sur notre conception générale de la connaissance n'ont peut-être pas encore été toutes tirées. Les procédures semblent être des objets qui circulent librement de l'action à la connaissance, de la théorie à la pratique, et qui permettent aussi bien de mobiliser le savoir pour atteindre des fins particulières, que d'augmenter notre capacité à découvrir, organiser, et transmettre de nouveaux savoirs.

Notre perspective explique que nous ayons formulé notre question directrice concernant la programmation en utilisant l'expression assez vague de *forme de connaissance* plutôt que par exemple *genre* ou *espèce de connaissance*. Nous voulons indiquer par là qu'il n'est pas fait l'hypothèse quant à la possibilité d'une taxonomie des savoirs, du type de celles que l'*Encyclopédie* a popularisées. La programmation semble bien être une compétence *transverse* aux disciplines établies (mathématiques, ingénierie, droit, médecine). Par ailleurs *genre* et *espèce* s'appliquent plutôt à des collections d'objets, tandis que *forme* semble plus approprié pour une activité.

À ce stade, il est nécessaire de clarifier l'usage que nous ferons des termes *connaissance*, *savoir* et *raison*. Concernant les deux premiers, nous tenterons de garder la distinction d'usage qui prévaut au moins depuis le dictionnaire de Lalande ([1926] 1992). La connaissance désigne d'abord l'*acte* d'avoir à l'esprit un objet de pensée, en tant qu'on le tient pour vrai ou réel, ou la compétence à accomplir cet acte. Le savoir, par contraste, désigne le contenu de cet objet de connaissance, s'il est suffisamment développé et systématisé[72]. Cependant, par commodité, nous pourrons aussi utiliser *connaissance*, surtout au pluriel, dans le sens de *savoir*.

Par *raison*, nous désignerons en général la faculté de connaître, en tant qu'elle procède de manière discursive, c'est-à-dire par l'argumentation. Là encore, nous adoptons un usage proche du sens commun, qui se passe de présupposés concer-

72. (Lalande [1926] 1992, vol. 2, p. 948).

nant les lois auxquelles devrait se soumettre la pensée discursive afin de prétendre à la connaissance. Il n'est donc pas question à ce stade de distinguer entre entendement, raison, et faculté de juger[73].

1.5 Situation dans le débat philosophique en général

L'extension de notre problématique que nous indiquons à la fin de la section précédente peut laisser perplexe. Elle peut paraître relever de l'infatuation naturelle du chercheur pour son sujet, qui a tendance à le présenter comme central et crucial pour l'élucidation de toutes sortes de questions. On pourrait douter en effet que l'étude de la programmation présente un intérêt épistémologique autre que régional, et qu'elle soit de nature à éclairer, voire à modifier, notre conception de la connaissance.

On peut d'abord répondre à cette perplexité en précisant notre champ d'études. Comme nous l'avons dit plus haut, nous ne prétendons pas dans ce travail donner un nouveau concept générique de la connaissance. Nous nous intéressons seulement au champ qui se situe dans les environs de ce qu'on appelle – de manière somme toute assez vague – *connaissance pratique* ou *technique*, dans leur rapport avec les mathématiques, la logique, et les théories scientifiques. Nous ne prétendons pas, par exemple, y réduire la connaissance scientifique elle-même, ni la connaissance historique ou narrative des faits humains, ou encore la connaissance esthétique, morale, etc. Tous ces domaines sont fermement hors de notre propos, même si, pour des raisons de commodité, nous utiliserons les termes de *connaissance* et *savoir* de manière non qualifiée, sans chaque fois rappeler ce cadre restreint dans lequel nous nous plaçons.

On pourrait également répondre à cette perplexité concernant l'importance de l'événement épistémologique que constituerait l'informatique en reprenant les faits que nous avons déjà présentés : l'importance des recherches et des hésitations autour la notion de *méthode* dès les Temps modernes, l'ampleur des changements qu'introduisent les ordinateurs dans notre monde humain, y compris dans nos entreprises de connaissance, l'hypothèse computationnelle de l'esprit, qui a orienté les débats de la philosophie pendant près d'un demi-siècle, la place prépondérante qu'acquiert l'informatique dans les cursus universitaires scientifiques, etc.

73. Nous reviendrons sur ces distinctions au chapitre 8.

Il nous semble plus intéressant ici de montrer l'importance qu'ont déjà prises les notions de *procédure* et de *programme* dans les développements contemporains des débats généraux que nous venons d'évoquer. L'opposition de la théorie et de la pratique n'est pas en effet une thématique qui parcourt seulement l'épistémologie (§ 15), elle irrigue de nombreuses strates de réflexion concernant la nature de la connaissance, en logique, en philosophie du langage ordinaire (§ 16), et enfin en science cognitive (§ 17). Cette mise en perspective large nous permet de préciser la spécificité de notre questionnement relativement à ces débats (§ 18), auxquels elle est cependant profondément intéressée, et auxquels elle pourrait contribuer en retour (§ 19).

§ 15. L'autonomie des techniques à l'égard des sciences

La question de la théorie et de la pratique est active dans le débat épistémologique concernant le rapport des sciences aux techniques. En particulier, l'autonomie du savoir technique à l'égard du savoir scientifique fut affirmée avec une force grandissante dans la seconde moitié du XX^e^ siècle, en réaction à l'« applicationnisme » triomphant, aux connotations positivistes, qui régnait dans l'après-guerre, et qui concevait, dans la lignée d'Auguste Comte, la technique comme la simple application du savoir scientifique à des problèmes pratiques[74].

En France, ce débat débute avec Georges Canguilhem, qui inspire le travail fondateur de Guillerme et Sebestik ([1968] 2007). Selon Canguilhem, chaque technique a sa logique de développement propre, ses problèmes, ses méthodes : elle peut solliciter les sciences sur certains points, dialoguer avec elles, mais ne saurait se réduire à l'application systématique de théorèmes scientifiques. Jan Sebestik, exposant ces idées, caractérise l'intelligence technique comme essentiellement constructive, par opposition à l'intelligence déductive de la science :

> Les arts et métiers se sont développés de manière autonome par rapport aux théories scientifiques. En particulier, l'invention technique est « une combinaison de solutions particulières efficaces ». Il faut donc renverser le rapport qui fait de la science l'origine des inventions. La technique industrielle ne relève pas de la science appliquée. Une machine n'est pas le résultat d'une simple application de la science, un « théorème solidifié ».

74. Sur l'applicationisme lié à Auguste Comte, voir (Deldicque 2022). Sur la conception dominante de la technique et de l'ingénierie comme subordonnées à la science dans les années 1950-1980, voir (Forman 2007).

> Pour [Canguilhem], la science est un savoir unifié et si possible organisé déductivement. Or, aucune déduction ne saurait conduire directement d'un ensemble de propositions à une construction, comme si le résultat de cette construction était à son tour un théorème. Il n'y a pas de déduction qui conduit de la science à ses applications ; ce passage suppose l'intervention de la technique, donc le franchissement de la ligne qui sépare le savoir théorique de l'activité de construction. Les sciences s'appliquent à travers la technique. Il est entendu que l'autonomie des techniques par rapport à la science n'exclut pas les influences et emprunts réciproques. L'initiative des recherches théoriques émane de la technique, ensuite un va-et-vient s'installe entre science et technique. Considérer la technique comme résultant de la science qui la précède ressort de la vision intellectualiste, dérivée, secondaire par rapport aux opérations du vivant pour assurer sa survie (Sebestik 2007, p. 124).

La réfutation du modèle « applicationniste » de la technique est aujourd'hui un fait bien établi par une multitude d'études montrant la complexité des pratiques des ingénieurs et de techniciens, pour lesquelles le savoir scientifique ne constitue qu'une faible inspiration. Mokyr (2016, p. 268-269) rappelle après d'autres que, alors même que le savoir scientifique était érigé aux XVI^e^ et XVII^e^ siècles en moteur de l'avancement des arts, un tel slogan est resté concrètement largement programmatique jusqu'au milieu du XVIII^e^ siècle. On sait par exemple que les machines à vapeur furent développées avant que les principes de la thermodynamique soient compris[75]. Mais cette indépendance de principe reste largement vérifiée aujourd'hui. Vincenti (1992) l'a montrée à l'œuvre en aéronautique, une technologie pourtant apparemment largement dépendante de la mécanique des fluides. On peut également citer, en épistémologie de la médecine, la notion de « ligne mentale » (*mindline*) qui caractériserait le savoir des praticiens, fait d'emprunts à de multiples sources dont les savoirs scientifiques ne seraient qu'une partie limitée[76].

Cependant, malgré cette autonomie des techniques à l'égard des sciences, il reste difficile de déterminer où se situerait la frontière caractéristique entre les unes et les autres. De nombreuses disciplines intermédiaires, comme les « sciences de l'ingénieur » ou les « sciences technologiques » sont venues brouiller l'opposition qu'on avait crue nette entre « sciences pures » et « techniques appliquées »[77]. Aussi Houkes (2009), qui réalise une revue des critères proposés par les philosophes pour distinguer le savoir scientifique du savoir technologique, conclut à

75. Voir la discussion et les références de (Meijers et Kroes 2013, p. 17-18) sur ce sujet.
76. (Gabbay et Le May 2010, 2016).
77. Voir les distinctions de (Niiniluoto 1993) à ce sujet.

son scepticisme sur la question. Par exemple, l'opposition entre la recherche de la vérité et celle de l'utilité, un critère souvent avancé, se révèle fragile à l'examen[78].

Ce brouillage de la distinction entre sciences et techniques est aujourd'hui bien identifié. Cependant son interprétation reste ouverte. La distinction est aujourd'hui souvent interprétée comme étant d'origine idéologique, historiquement datée. Dans une collection d'études sur l'invention technique aux Temps modernes, les éditeurs justifient ainsi leur demande aux auteurs d'éviter les termes *science* et *technique* autant que possible[79]. Le terme *technoscience*, courant aujourd'hui, véhicule cet état d'esprit. L'historien des sciences Paul Forman note, qu'à nos yeux postmodernes, *technologie* dénote également la *science*, tandis que, pour les modernes (avant 1980, dans sa définition), le contraire était vrai[80].

L'informatique vient renforcer ces deux thèses apparemment contradictoires, d'une part l'affirmation de l'autonomie des techniques à l'égard des sciences, et de l'autre, l'impossibilité de les distinguer nettement. Concernant le premier point, nous avons déjà insisté[81] sur le fait qu'en informatique, la théorie semblait se réduire à l'étude de la pratique.

Concernant le second point, l'informatisation généralisée de l'ingénierie rend celle-ci plus « scientifique » en ce qu'elle a de plus en plus recours à la simulation, qui s'appuie sur des lois physiques et des outils mathématiques. Les sciences de l'ingénieur et les sciences technologiques construisent ce pont entre science et ingénierie par la préparation de *modèles computationnels* adéquats[82]. Les méthodes statistiques sont aussi de plus en plus efficaces à imiter le savoir-faire des ingénieurs et des techniciens dans leur travail de conception[83].

De plus, la science elle-même est transformée par l'informatique, et cela au moins de deux manières. D'abord, le début des années 1980 voit l'éclosion, au sein de diverses sciences, d'approches dites « computationnelles », en ce qu'elles reconceptualisent leurs objets à partir de concepts informatiques, et notamment celui d'*information*. Ainsi, le prix Nobel David Baltimore, un biologiste, argumenta dès les années 1990 que la biologie devait être redéfinie comme l'étude des processus

78. Nous l'examinons nous-même, ainsi que toute cette discussion, plus loin dans ce travail (§ 139).
79. (Roberts, Schaffer et Dear 2007, p. xvi).
80. (Forman 2007, p. 4).
81. Voir plus haut, § 12.
82. (Boon et Knuuttila 2009).
83. (Regenwetter, Nobari et Ahmed 2022).

informationnels sous-tendant le fonctionnement des cellules et plus généralement tous les phénomènes du vivant[84]. De nombreuses autres sciences ont ainsi développé des branches « computationnelles », comme la chimie ou l'astrophysique. D'autre part, l'usage intensif des méthodes numériques, permis par l'informatique, permet d'établir des prédictions sans compréhension structurelle, c'est-à-dire théorique des phénomènes concernés. Ainsi, Napoletani, Panza et Struppa (2011) avancent l'idée d'une *science agnostique*, qu'ils caractérisent ainsi :

> Loin d'être le résultat d'une incapacité momentanée à créer des isomorphismes entre phénomènes et théories, ce paradigme relève d'une nouvelle conception qui s'impose peu à peu et qui mérite d'être comprise et appréciée comme telle. Au lieu d'essayer de comprendre et de modéliser un phénomène, ce paradigme suggère qu'un scientifique doit aborder un phénomène avec un ensemble limité d'hypothèses, et doit rechercher des techniques spécifiques capables de résoudre certains des problèmes qu'il présente, sans tenter aucune sorte de compréhension structurelle du phénomène lui-même[85].

Une telle caractérisation de la science la rapproche étonnement de la conception traditionnelle de la technique, comme savoir permettant de produire les effets sans connaître les principes. La notion de savoir *théorique* est elle-même remise en cause.

Toutes ces convergences que suscite l'informatique rendent la distinction entre sciences et techniques de plus en plus problématique, et par là même remet en cause les cadres traditionnels de l'épistémologie.

§ 16. Logiques de l'action, formes de vie

Une deuxième strate du débat se situe au niveau logique, et porte sur la possibilité de distinguer savoirs théoriques et pratiques par des différences de forme. En particulier, les savoirs pratiques sont souvent dits être prescriptifs, et donc s'énoncer par des impératifs : cette idée se trouve déjà chez Kant, pour qui les impératifs donnent la forme des principes de la philosophie pratique. Von Wright montre, dans des travaux sur lesquels nous reviendrons, l'impossibilité de réduire le syllogisme pratique (« *Qui veut chauffer sa cabane doit faire un feu / X veut chauffer sa cabane / X fait un feu* ») à un syllogisme théorique[86]. Dans les années 1960,

84. (Baltimore 2001).
85. (Napoletani, Panza et Struppa 2011, p. 3).
86. Voir plus loin § 138.

à la suite du développement des logiques déontiques, de nombreux systèmes de « logiques des commandes » voient le jour[87]. Prenant une position contraire, Herbert Simon, avec d'autres, argumente que la logique propositionnelle classique est suffisante pour exprimer tous les savoirs pratiques[88].

L'informatique est utile pour éclairer ce débat. En particulier, certaines méthodes de vérification formelle des programmes ont été à l'origine de la logique dynamique[89], considérée aujourd'hui comme la plus intéressante pour analyser l'action et le raisonnement pratique[90]. Par ailleurs, l'informatique permet de distinguer les oppositions superficielles entre description et prescription et celles qui sont structurelles. Par exemple, on peut énoncer l'algorithme d'Euclide pour trouver le plus grand diviseur commun de deux entiers de manière prescriptive, mais l'action qu'il dirige ne fait que parcourir une relation mathématique récursive descriptible par de simples propositions. Par contre, une logique intrinsèquement prescriptive est requise dès qu'il faut réguler l'accès à des ressources partagées[91]. Ces questions renvoient à d'autres équivalences, bien connues des informaticiens, mais dont on commence seulement à apprécier l'intérêt philosophique, comme celles entre donnée et algorithme[92], entre fichier et programme[93], entre structure et processus. Tous ces éléments sont essentiels, nous semble-t-il, pour le renouvellement de notre compréhension de l'opposition entre connaissance théorique et pratique.

L'idée d'une incompatibilité de forme entre les deux types de savoirs peut être fondée sur l'idée, plus radicale, que le savoir pratique est d'abord non-propositionnel, qu'il est avant tout une compétence et une disposition, et qu'en ce sens il précède et fonde tout savoir théorique. Ryle ainsi propose la distinction, devenue célèbre, entre *savoir* et *savoir-faire*, entre *know-that* et *know-how*. Ici, la ligne de fracture entre connaissance théorique et connaissance pratique ne passe pas tant entre science et technique, qu'entre savoir propositionnel et compétence incarnée. On sait faire de la bicyclette, ou faire la cuisine, sans nécessairement pouvoir nécessairement l'exprimer avec des mots.

87. Voir par exemple (Rescher 1966).
88. Pour un historique des logiques impératives, voir (Hansen 2008).
89. (Pratt 1990).
90. (Zwart, Franssen et Kroes 2018). Nous étudions ce point au cours de notre travail, § 138.
91. Ces points seront étudiés au chapitre 12.
92. (Pégny à paraître, p. 219).
93. (Mélès 2022).

Bien au contraire, l'explicitation des méthodes est dérivée de ce savoir pratique ; il y a donc tout un pan de notre connaissance propositionnelle qui en dépend, bien plus qu'il ne dépend de principes théoriques :

> Lorsqu'une personne sait comment faire certaines choses, sa performance est d'une certaine manière gouvernée par des principes, des règles, des canons, des standards ou des critères […] Il est toujours possible en principe, sinon en pratique, d'expliquer pourquoi elle tend à réussir, c'est-à-dire, à énoncer les raisons de ses actions. […] Mais son observance de règles, principes, etc. doit être – si elle est jamais présente – réalisée dans la réalisation de ses tâches. Celle-ci ne doit pas (quoiqu'elle le puisse) être également exposée dans une autre réalisation, qui serait d'ânonner, intérieurement ou extérieurement, ces règles ou principes. On doit travailler judicieusement ; on peut aussi exposer des jugements. […] Juger (ou penser propositionnellement) est l'une (mais seulement une) des manières d'exercer son jugement ou de trahir sa bêtise […] En somme l'acceptation propositionnelle des règles, raisons ou principe n'est pas mère de leur application intelligente ; elle en est au contraire la belle-fille[94].

Ce qu'affirme Ryle ici, ce n'est pas seulement que les gens savent faire des choses sans pouvoir dire comment ils font ; car cela est évident s'il s'agit de toute notre activité réflexe, comme respirer, marcher, etc. Ce qu'il affirme, c'est que les gens agissent *raisonnablement* sans passer toujours par une explicitation langagière – qu'il y a des manifestations de la raison qui ne passent pas par des raisonnements propositionnels. Au contraire, dit-il, ceux-ci ne sont eux-mêmes qu'une manifestation très particulière de l'intelligence, un moyen que l'on se donne, dans certains cas, pour assurer la réussite de l'action. Connaître, c'est d'abord savoir faire quelque chose, et seulement, de manière dérivée, savoir quels sont les faits et les raisons pour lesquels cette action réussit.

> Connaître est un verbe capacitaire, et un verbe capacitaire de cette sorte spéciale utilisée pour signifier que la personne ainsi décrite peut amener les choses à réalisation, ou les faire correctement[95].

Cela veut dire que *connaître* ne dénote pas une compétence spéciale de l'esprit, mais qu'il est un verbe générique pour désigner une large gamme de compétences :

> Des verbes de disposition comme connaître, croire, aspirer […] signifient des compétences, des tendances ou des inclinaisons à faire, non pas des choses d'une seule espèce, mais de plein d'espèces différentes[96].

94. (Ryle 1945, p. 9).
95. (Ryle [1949] 2002, p. 133).
96. (ibid., p. 118).

La conséquence de ceci, c'est que la connaissance ne peut être le fait d'une raison surplombante et abstraite, dans le langage de laquelle elle pourrait être tout entière décrite. Chaque compétence – savoir nager, calculer, connaître les convenances, avoir de l'esprit – s'inscrit dans une réalité et un contexte spécifiques qui lui confèrent sa signification. Savoir nager suppose qu'il y ait des étendues d'eau, les convenances qu'on se situe dans un certain milieu social, le calcul qu'on dispose d'un concept de nombre, etc.

Ce caractère intrinsèquement multiple de la connaissance, ancré, dans une multiplicité de réalités situées évoque les formes de vie de Wittgenstein. Ryle n'avait sans doute pas connaissance de cette notion lorsqu'il publia *Le Concept de l'Esprit* en 1949, puisqu'elle n'apparaît que dans les *Investigations Philosophiques*, cependant Ryle et Wittgenstein avaient partagé de nombreuses idées, notamment concernant l'idée de *dispositions* à la fin des années 1930[97].

En tous les cas Wittgenstein affirme une idée similaire à celle de Ryle, lorsqu'il montre qu'on ne peut parler de connaissances, de significations, de jugements, de formes logiques ou de paroles sans recourir, d'une manière ou d'une autre, à leur ancrage dans la vie humaine, dans toute sa diversité – ce qui ne veut pas dire qu'il soit possible de décrire positivement ces formes de vie, de les dénombrer, de les classer[98] :

> Mais combien existe-t-il de catégories de phrases ? L'assertion, l'interrogation et l'ordre peut-être ? — Il y en a d'innombrables, il y a d'innombrables catégories d'emplois différents de ce que nous nommons « signes », « mots », « phrases ». Et cette diversité n'est rien de fixe, rien de donné une fois pour toutes. Au contraire, de nouveaux types de langage, de nouveaux jeux de langage pourrions-nous dire, voient le jour, tandis que d'autres vieillissent et tombent dans l'oubli. (Les changements en mathématiques pourraient nous donner une image approximative de cette situation.)
>
> L'expression « jeu de langage » doit ici faire ressortir que parler un langage fait partie d'une activité, ou d'une forme de vie[99].

L'intérêt spécifique de Wittgenstein pour notre propos est que, contrairement à Ryle, il accorde une bien plus grande importance au caractère régulatoire du langage, c'est-à-dire à sa fonction d'énoncer des règles qui dirigent aussi bien notre

97. Concernant l'influence de Wittgenstein sur Ryle, voir (Place 1999, p. 364).
98. (C. Martin 2018, p. 2).
99. Wittgenstein, *Philosophische Untersuchungen*, §23. Traduction française : (Wittgenstein [1953] 2014).

compréhension de la situation que notre comportement face à elle. Les règles sont souvent constitutives de nos dispositions, et non leur expression tardive et dérivée – sujet sur lequel la position de Ryle nous semble rester ambiguë. Ainsi, tout ce qui a le caractère d'un jeu suppose des règles et des procédures symboliques qui *signifient* ce que veut dire savoir y jouer. Bien plus que cela, selon Juliet Floyd, les procédures sont les formes élémentaires par lesquelles nous enchaînons nos pensées et notre appréhension du monde :

> L'enchaînement (*chaining*) [de pensées] rend manifeste des possibilités et des nécessités de mouvement et de non-mouvement, de raccourcissement et de distribution, de réarrangement de liens, la possibilité de tracer [des voies] de bout en bout, en arrière et en avant. Il s'agit d'une synthèse qui tient ensemble un cours de pensée qui peut aussi être analysée en parties – substituables cependant par d'autres parties, d'autres liens et enchaînements, d'autres synthèses, etc. [...] Nous sommes des relieurs, travaillant sur une toile de fond universelle, un tourbillon de vie dans lequel des procédures, des routines, des enchaînements, se sont déjà imbriqués. La logique s'occupe d'établir et d'imbriquer ensemble des procédures et des possibilités de procédures, d'en sonder les limites[100].

Les règles et les procédures ne seraient donc pas des objets abstraits, détachés de la vie, mais la logique même de nos formes de vie, les formes logiques de la vie, comme elle le souligne encore :

> L'accomplissement philosophique ici est une conception dynamique et procédurale du logique, de la forme logique comme quelque chose *dans* et *de* la vie elle-même. Il faut souligner que ce n'est que dans sa pensée mature que Wittgenstein réussit à ancrer la logique *dans* la vie, en tant que quelque chose *de* la vie, en la rendant omniprésente par le biais de critères et de procédures de notre vie quotidienne[101].

Pour J. Floyd, l'influence d'Alan Turing, le célèbre mathématicien, fut décisive dans la formation de ce concept de *forme de vie*. Turing et Wittgenstein, par leurs nombreuses conversations à Cambridge, eurent selon elle une influence décisive l'un sur l'autre. D'une part, Turing aborde la notion de calcul non pas formellement, comme le fait Church, mais par l'analyse *de ce qu'on fait lorsqu'on calcule*, ce qui est une démarche typiquement wittgensteinienne. D'autre part, l'article de Turing de 1936 aurait été décisif pour Wittgenstein à plusieurs titres. Il lui aurait d'abord donné la définition de ce qu'est une procédure, de ce que signifie *suivre une règle*. Ensuite, en prouvant qu'il n'existe pas de procédure qui permette, au sein d'un

100. (J. Floyd 2016, p. 40).
101. (J. Floyd 2018, p. 68).

système formel donné, de déterminer quelles procédures sont valides ou non, il lui aurait suggéré l'idée qu'une logique universelle, surplombant toutes les formes de vie, doit être abandonnée, et qu'il faut concevoir au contraire les systèmes logiques à partir de celles-ci.

§ 17. Incarnation et outillages de la pensée

Une quatrième strate du débat se situe au niveau de l'anthropologie de la connaissance. Depuis une trentaine d'années, plusieurs courants de science cognitive, qu'on regroupe souvent sous la dénomination « 4E » pour signifier leur proximité d'intention, insistent sur les caractères incarné (*embodied*), situé (*embedded*), énactif (*enactive*) et étendu (*extended*) de la connaissance. Il ne peut s'agir ici bien sûr de décrire les différentes thèses et débats qui réunissent et opposent ces réflexions : nous renvoyons là-dessus à la synthèse, déjà un peu vieille, de Clark (2010). Nous relevons trois points qui permettent de mettre ces débats en relation avec notre sujet.

(1) Tous ces courants s'accordent dans leur opposition à l'hypothèse computationnelle de l'esprit, qui conçoit la pensée comme manipulation réglée de représentations (ou encore de symboles, informations, données) dont les composantes élémentaires seraient données par le biais d'interfaces perceptives entre l'organisme et son environnement (les « données des sens »). Au contraire, la perception doit être comprise comme « action perceptivement guidée », c'est-à-dire identifiée à la structure sensorimotrice dynamique de l'agent, qui détermine *quelle est la situation* dans laquelle il se trouve – c'est-à-dire ce qui est pertinent pour guider son action, et cela sur plusieurs strates de sens – sensoriel, endocrinal, émotionnel, social. La situation co-évolue donc avec l'agent. Les structures cognitives – les concepts, les règles – émergent de la récurrence de ces schémas sensori-moteurs[102].

(2) En remettant ainsi en cause l'hypothèse de l'indifférence des processus cognitifs à leur réalisation corporelle et à leur contexte, une autre hypothèse naturelle de la théorie classique de la psychologie cognitive devient également problématique, à savoir qu'ils soient réalisés dans les bornes de la « matière cérébrale ». La frontière de la « boîte crânienne » apparaît en effet arbitraire aux nouveaux cognitivistes. On montre par exemple que l'espace physique est utilisé activement pour faciliter la perception, le calcul, la prise de décision, en regroupant mentalement

102. (Varela, E. Thompson et Rosch [1991] 2016, p. 173-178).

ou physiquement les objets en jeu [103]. L'idée que la perception est une réception passive de données en « entrée » par le cerveau, qui les transforme selon des règles en « sorties » envoyés aux nerfs moteurs – ce modèle *séquentiel* de la perception – est rejeté : la théorie sensorimotrice de la perception avance au contraire que toute perception est déjà active, qu'elle oriente le corps et les gestes pour établir une synthèse visuelle ou sonore de l'objet, qu'elle pré-remplit souvent cette synthèse d'anticipations schématiques, et qu'elle se corrige progressivement dans une boucle itérative que forment le corps et l'environnement, où le système nerveux n'est qu'une partie du processus à l'œuvre.

Ainsi, si la connaissance est constituée de boucles itératives impliquant indissociablement l'organisme et son environnement, alors elle est aussi nécessairement *historique* : car ces boucles forment des habitudes, des automatismes, mais également se construisent les unes à partir des autres, et ce sont elles qui, ultimement permettent d'expliquer l'intelligence :

> Ces étonnantes montagnes de compétences, créées et maintenues délibérément, incluent souvent une large variété d'actions épistémiques[104]. De telles actions, profondément enfouies dans les systèmes d'habitudes imbriqués qui permettent à l'agent incarné et situé, dans un tourbillon d'engagements actifs avec toute sorte de béquilles, d'outils et d'artefacts, de réaliser jusqu'aux tâches les plus exigeantes cognitivement[105].

Cette historicité implique que l'environnement n'est pas seulement présent dans l'acte immédiat de la connaissance, mais qu'il façonne et innerve nos processus cognitifs les plus intimes. Ainsi, les compétences cognitives d'un comptable dépendent de l'organisation de son espace de travail (rangement de ses dossiers, structure visuelle des pages) – et cet espace est en retour façonné par des optimisations, conscientes ou inconscientes, réalisées au cours du temps pour augmenter leur efficacité[106]. Plus généralement, c'est tout notre environnement socio-culturel qui entre au sein nos processus cognitifs, et notamment nos interactions avec autrui, la situation sociale dans laquelle nous nous trouvons, ou encore le rôle professionnel qui nous est assigné.

(3) La thèse de la « cognition étendue » permet de concevoir la technique comme *constituante* de nos capacités d'action et de connaissance. Elle s'accorde

103. Voir (Clark 2010, p. 64-66) pour des exemples et des références.
104. Les actions épistémiques sont, pour Andy Clark, des actions visant à acquérir de l'information.
105. (Clark 2010, p. 75).
106. (ibid., p. 68).

par là avec la thèse de la *projection d'organes*, formulée initialement par le philosophe allemand Kapp et développée (entre autres) par l'anthropologue Leroi-Gourhan, selon laquelle le développement de l'être humain s'est fait par extériorisation progressive de ses possibilités corporelles – déchirer avec les incisives, racler avec les ongles, attraper avec la main, marcher, etc. – dans des outils qui les amplifient. Ainsi Leroi-Gourhan, sans doute inspiré par les idées cybernétiques[107], présente l'informatique comme l'étape ultime de projection des fonctions corporelles humaines en outils, prenant en charge la mémorisation et la transmission de ses « chaînes opératoires » (ses gestes techniques)[108].

Le philosophe Pierre Steiner présente ainsi la thèse de la technique comme constituante :

> L'objet technique joue un rôle constituant pour nos capacités d'action, de raisonnement ou encore de perception, en étant non-perçu, ou encore transparent (je perçois par mes lunettes ; je ne perçois pas mes lunettes). À partir de Don Ihde, on peut, au sein de l'objet technique constituant, distinguer la constitution se réalisant par incorporation (l'usager fait l'expérience de l'objet technique comme partie de lui-même ; l'objet est ainsi une extension transparente du corps propre (moyennant appropriation), amplifiant ses pouvoirs d'action et de perception) de la constitution se réalisant sur un mode herméneutique (l'objet technique médiatise mon accès et ma relation à un nouveau monde autrement inaccessible, en m'offrant quelque chose de nouveau à voir, souvent par le biais de représentations à déchiffrer (produites par exemple par le télescope, le microscope, le thermomètre, la TEP, le sismographe,…)).

En particulier, l'écriture est une technologie cognitive essentielle :

> L'écriture nous permet de spatialiser et d'objectiver nos pensées et nos discours, d'en faire des objets de critique et de partage, en les soustrayant au flux temporel de l'oralité et de la pensée. Elle permet également un accroissement et un enrichissement du savoir en permettant de le stocker, de l'accumuler. L'écriture rend également et surtout possible, chez Goody (1977), un nouveau type de rationalité : une raison graphique. Cette raison graphique entraîne notamment le développement de nouvelles compétences de perception et de compréhension (tableaux, listes, formules,…) [et amène] un développement des attitudes cognitives critiques et sceptiques envers le texte, mais aussi de la pensée logique (syllogismes)[109].

107. (Schlanger 2023, p. 326).
108. (Leroi-Gourhan 1965, t. 2, p. 53-56).
109. (Steiner 2010, p. 22).

Suivant l'interprétation de l'*Origine de la géométrie* de Husserl par Derrida, reprise par Stiegler, Steiner avance ainsi que l'inscription graphique qu'est l'écriture permet l'advenue des idéalités de la logique, des mathématiques et donc de la science.

Bachimont (2004) et Lassègue et Longo (2012) s'inspirent de Goody pour présenter l'informatique comme une nouvelle révolution au sein de nos techniques cognitives, comme une « révolution graphique ». Bachimont avance que l'informatique rend accessible à la réflexion de nouvelles structures mentales, comme les réseaux ou les niveaux d'abstraction. Il rejoint par là, en une certaine mesure, la pensée computationnelle[110]. Lassègue insiste plutôt sur la désymbolisation qu'elle entraîne, « différ[ant] la compréhension de la signification à une étape ultérieure et extérieure au traitement informatique lui-même[111] » et par là créant une autonomie et une opacité de cette forme d'écriture à l'égard des agents humains. L'écriture informatique n'est pas dénuée de sens, mais est « une autre manière de produire du sens », ce qui bouleverse en profondeur le rapport de l'être humain au monde, instaurant un nouvel « ordre graphique ».

§ 18. Procédures, techniques et savoir-faire

Les débats contemporains que nous venons d'évoquer éclairent chacun d'une perspective différente l'opposition entre théorie et pratique, soit qu'on les considère dans leur cohabitation, soit dans leur prétention exclusive à fonder la connaissance. Si chacune de ces discussions place la ligne de partage entre ces deux pôles à des endroits différents (entre technique et science, entre compétence et représentation discursive, entre discours impératif ou déclaratif, etc.) il y a toujours, à l'arrière-plan, le questionnement concernant la nature de la connaissance et son modèle traditionnel, qu'on peut appeler après John Dewey la *théorie de la connaissance - spectacle* (*the spectator theory of knowledge*), et qui la conçoit comme observation ou vision de la chose mise à distance, plutôt que comme manipulation et interaction avec elle.

Dans le développement suivant, nous montrerons pourquoi notre travail ne s'inscrit cependant pas directement dans l'un ou l'autre de ces débats, même s'il s'en inspire intimement. Nous avons déjà montré qu'il n'était pas évident de présenter la programmation comme le simple prolongement d'une raison procédurale

110. Voir plus haut, § 10.
111. (Garapon et Lassègue 2018, p. 46).

qui serait déjà à l'œuvre dans les savoir-faire, les techniques et d'une manière générale toutes les réflexions pratiques[112]. Dans ce développement, nous mettons en évidence plusieurs différences entre la programmation d'une part, et les techniques et savoir-faire de l'autre. D'abord, leurs domaines ne se recouvrent que partiellement ; ensuite ce qu'on appelle *règle* dans les deux cas ne signifie pas de manière évidente la même chose ; enfin et surtout, la programmation a tout simplement trait à des *machines*, et non à la direction de l'action humaine.

(1) Il est aisé de remarquer que connaissance technique et connaissance des procédures ne se recouvrent que partiellement. D'une part, la première ne porte pas seulement sur des procédures, elle comprend également la connaissance des matériaux et des outils, ainsi que de leurs propriétés ; elle s'appuie sur des modèles et des traditions. Elle tient compte de normes de qualité, de sécurité, écologiques, etc. Enfin, elle s'exprime souvent par des savoir-faire et par des conventions tacites transmis par apprentissage plutôt que par des méthodes explicites.

D'autre part, en sens inverse, le domaine des procédures effectives dépasse largement ce qu'il est convenu en général d'appeler technique ou technologie. Ces procédures peuvent être appliquées au domaine du droit, des affaires, de la guerre, de la vie courante, du divertissement et de la culture. Surtout, elles concernent de manière originaire les domaines des mathématiques, du langage et de la logique, dont on pourrait certes rappeler qu'ils constituaient les « arts » libéraux dans les cursus anciens, mais qui sont en général considérés comme partie, voire comme fondements, de la connaissance théorique.

(2) Notre questionnement ne porte pas non plus sur les compétences incarnées, les savoir-faire, ou les usages inscrits dans les « formes de vie » wittgensteiniennes. Il est trop souvent considéré par les auteurs cités plus haut que la mise en procédure de savoir-faire va sans dire et ne pose pas de problème particulier. Ainsi, Bachimont écrit :

> La tendance naturelle de la méthode est de poursuivre sa formalisation de manière à pouvoir fonctionner comme un programme et donc à mobiliser l'agent humain non comme un agent interprétant et improvisant mais comme un simple exécutant. Parfois, la méthode élimine la mobilisation de la médiation interprétative en s'objectivant complètement en un dispositif technique et machinique, comme on le voit dans les systèmes de gestion et l'industrialisation de nos techniques cognitives[113].

112. Voir plus haut, § 13.
113. (Bachimont 2010, p. 64).

De la même manière, pour J. Floyd, la mise en procédure de tâches s'inscrit dans l'activité fondamentale de *relier* les choses les unes aux autres dans des chaînes de significations, d'activités et de liens sociaux :

> Nous sommes des *relieurs* : de mots et de phrases, de procédures, de routines et d'activités, de nous-mêmes les uns avec les autres, de pensées, de mots et d'actions. Nous assemblons des éléments dans des arrangements qui tiennent. Nous sommes capables de nous lier à des procédures et à des routines[114].

Ailleurs dans le même article, elle présente les procédures formelles des machines de Turing comme la forme extrême de cette activité ordinaire de mise en procédure :

> Les procédures symboliques séquentielles manifestent des nécessités, mais celles-ci ne dépendent pas et ne peuvent pas dépendre d'une procéduralisation surplombante de la pensée. Au cœur de l'analyse [de Turing] se trouve plutôt l'idée schématique de la procéduralisation, de l'amalgame de routines, en tant que telles[115].

Ces présentations insistent sur la continuité qui pourrait exister entre les savoir-faire et les procédures formelles, provenant d'une même tendance à organiser et « rationaliser » nos activités dans l'expérience. Bachimont parle ainsi d'une *tendance naturelle* des méthodes à se transformer en programmes, et Floyd de l'activité fondamentale de *lier ensemble* nos actions en routines et en procédures. Il est tentant de postuler une telle continuité, car elle permet d'inscrire la programmation informatique dans une compétence épistémique bien plus large, voire dans notre être-au-monde élémentaire – ce qui est le sujet même de notre questionnement.

Cependant, nous ne pouvons dès l'abord postuler qu'une telle continuité va de soi. Traduire un savoir-faire dans une procédure effective exige de fixer de manière univoque le langage des actions élémentaires qui la composent : quelles elles sont, quelles sont leurs conditions, ce qu'elles produisent comme résultat, par exemple. Or cette transmutation n'est pas inoffensive. Lorsque nous *décrivons* ainsi nos pratiques dans un langage formel, nous ne faisons pas que supprimer la part d'interprétation possible de chaque étape de la méthode. Nous nous donnons le moyen, non seulement de préciser notre savoir-faire empirique, mais également de déployer des raisonnements systématiques sur ces éléments afin de *trouver* la solution d'un problème. Il suffit, en effet, d'énumérer toutes les combinaisons possibles du système jusqu'à ce faire. On objectera qu'une une telle démarche n'est pas

114. (J. Floyd 2016, p. 70).
115. (ibid., p. 16).

vraiment une méthode, mais la solution du pire, à laquelle on se range lorsqu'on est justement démuni d'aucun savoir-faire : explorer systématiquement toutes les options possibles. Par ailleurs, une telle méthode ne fonctionne plus nécessairement dès qu'on a affaire à un espace infini, fut-il aussi simple que celui des entiers naturels, puisqu'il y existe des problèmes indécidables. La programmation est justement la réponse à ces objections : dans un espace systématique, il devient possible de *programmer* la recherche afin qu'elle trouve une solution rapidement, en évitant de tourner sur elle-même. Ainsi par exemple, lorsqu'on est dans un labyrinthe, l'algorithme de Trémeaux exige certes d'explorer systématiquement les issues qui se présentent. Mais en exigeant qu'on *marque* chaque couloir traversé de façon à ne jamais l'emprunter deux fois dans le même sens, il permet de réduire progressivement le nombre d'options envisageables et garantit de sortir du labyrinthe à coup sûr.

Il y a donc une différence d'essence, et non de degré, entre les règles d'un langage formel et les règles qui guident notre action habituelle, ce que Ryle vit très bien lorsqu'il proposa de distinguer règles canoniques et règles « procrustéennes[116] ». Les premières sont les règles communes, similaires aux règles de l'art, qui indiquent dans quelles directions on doit chercher une bonne solution, mais laissent une large part au jugement. Parmi elles on trouve, par exemple, les règles de l'équitation, de la rhétorique, de la navigation. Les secondes, au contraire, sont tranchantes, en général binaires, et permettent de juger immédiatement si une suggestion est conforme à la règle ou non. De ce type sont les règles de la grammaire, de la logique, des jeux en général.

Cette distinction se comprend aisément par l'exemple du jeu d'échecs :

> Le joueur ne joue tout simplement pas aux échecs s'il n'observe pas les règles procrustéennes du jeu, mais il ne joue pas efficacement s'il n'applique pas également des maximes de stratège[117].

À la différence donc des règles procrustéennes, qui sont en quelque sorte constitutives de l'effort même en question, les canons sont des conseils et des maximes basés sur l'expérience, qui sont pertinents la plupart du temps, mais qui éludent toute tentative de systématisation – Ryle emploie ici, de manière éclairante, le terme *co-*

116. On se rappelle que Procruste était un brigand de l'Attique qui, après avoir invité les voyageurs à dormir chez lui, sciait les membres de ceux qui étaient trop grands pour le lit qu'il leur offrait, et les élongeait lorsqu'ils étaient trop petits.

117. (Ryle [1971] 2009, p. 240).

dification :

> Les règles canoniques [...] résistent communément à la codification. [...] On les apprend par la pratique et non par le dressage. Elles sont enseignées par la critique plutôt que par cœur, et nous n'avons jamais fini de les apprendre.

La programmation est ainsi procrustéenne au sens de Ryle, et se situe donc du côté de la logique et de l'arithmétique, se distinguant radicalement de l'ingénierie et du droit, où le formalisme est, tôt ou tard, complété par l'acte de jugement, qui ne peut être réduit à une règle procrustéenne. Ainsi écrit-il :

> Toutes les méthodes, techniques, arts, compétences, etc. sont sujets à des règles canoniques[118].

Cela ne veut pas dire que la programmation et les mathématiques en général n'ont pas leurs propres canons ; Ryle insiste sur l'élégance et l'à-propos qui guident les mathématiciens. Mais leur objet même est de codifier des règles procrustéennes. Bien entendu, Ryle pense que la réduction procédurale des méthodes est impossible, et qu'elle manque même leur aspect essentiel puisque connaître une méthode, à proprement parler, c'est savoir *quand* l'appliquer, et *comment*, ce qui exige immédiatement une interprétation[119].

Nous n'affirmons pas ici l'incommensurabilité absolue des règles canoniques et des règles procrustéennes, comme le fait Ryle. Nous voulons seulement montrer, à ce stade introductif de notre travail, que leur identité ou leur continuité d'essence n'est pas évidente à postuler.

(3) Enfin, la dernière différence entre programmation et savoir-faire est qu'elle concerne des ordinateurs, et non des êtres humains. Cette différence est souvent minimisée. On se représente qu'élaborer un plan d'action précis, puis le suivre à la lettre, c'est « se programmer soi-même ». On dit aussi parfois d'une personne qu'*« elle était programmée »* à faire telle ou telle chose, notamment en évoquant l'influence de son éducation sur sa carrière, ou encore d'un animal qu'il est *« programmé pour tel comportement »*, en référence à son *programme génétique* ou à son conditionnement. De telles expressions métaphoriques sont récentes, et certainement contaminées par l'informatique. Elles s'ancrent cependant dans des métaphores plus anciennes. On parle en effet d'un comportement *machinal* ou *automatique* – termes qui datent du milieu du XVIII^e^ siècle[120]. Ces métaphores semblent

118. (ibid., p. 241).
119. (ibid., p. 242).
120. Voir les articles correspondants du *Grand Robert de la Langue Française* (ROBERT et REY 1986).

d'une certaine manière légitimes : ne se forge-t-on pas des habitudes en répétant consciemment des séquences planifiées de gestes, de manière à les incorporer progressivement, jusqu'à ce qu'on les exécute « sans même y penser », c'est-à-dire comme une machine ?

Cette idée peut être justifiée à l'aide de la thèse de la technique comme constituante, évoquée plus haut[121], par l'établissement d'une analogie stricte entre les inscriptions externes et internes de la pensée. Aussi Bachimont (2010, p. 138) n'hésite-t-il pas à écrire :

> La conscience est […] une dynamique au confluent de deux systèmes techniques : le système technique vivant et privé qu'est le corps propre, le système technique des outils et instruments extérieurs au corps. […] Ainsi, écouter un enregistrement sonore, c'est en fait le retranscrire et le mémoriser en son corps propre. L'enregistrement externe, le compact-disc par exemple, propose une appréhension, une répétition et une sélection que la conscience, à partir de son intériorité structurée par le corps propre, appréhende, reproduit et sélectionne également.

On retrouve ici des analogies qui rappellent l'hypothèse computationnelle de l'esprit. Bachimont affirme même que « la pensée est une action d'inscription sur le corps propre comme support. Penser c'est s'écrire, ou encore se ré-écrire[122]. »

De telles identifications doivent à notre avis être prudemment mises en suspens. Nous avons déjà évoqué plus haut la surprise qu'éprouvent les programmeurs à la difficulté de la programmation, au type d'erreurs qu'on y commet, et qui pointent vers l'existence de différences majeures entre un programme informatique et un plan d'action destiné à guider des êtres humains. Les contextes d'arrière-plan nécessaires à l'exécution de l'un ou de l'autre sont entièrement distincts – il n'y a pas besoin en général d'un langage formel pour décrire un plan d'action. Son exécution est au contraire chaque fois sujet à une interprétation, qu'elle soit consciente ou non. Lorsqu'on *apprend* une nouvelle procédure (une recette de cuisine, un geste technique ou sportif, etc.) une attention supérieure commande nos actions, qui peut se diriger vers des détails ou des aspects délicats du geste comme, par exemple, mélanger la sauce à une certaine vitesse, ou délier la jambe gauche avant de sauter, etc. La procédure humaine est tout entière adaptée aux gestes déjà acquis par son exécutant, elle ne se conçoit que de manière différentielle. Comme le dit Ryle, nous *accompagnons* notre performance de préceptes et d'admonitions,

121. Voir plus haut, § 17.
122. (Bachimont 2010, p. 125).

ce qui veut dire tout autre chose qu'exécuter à la lettre une série d'instructions formelles.

§ 19. Vérité et effectivité

La particularité de notre questionnement relativement aux débats contemporains rappelés plus haut est donc celle-ci. Sans nier l'ancrage de la raison pratique dans les savoir-faire, les formes de vie, ou l'expérience incarnée du monde, il semble y avoir, avec la programmation, une *rupture épistémique* qui consiste à dégager l'art de construire des procédures dans son autonomie. Cette rupture semble pouvoir être comparée à celle qu'introduit la géométrie relativement aux arts pratiques de l'arpentage et de la construction, qui lui permet de n'être pas une simple abstraction de ces pratiques, mais de dégager un champ propre de significations, de problèmes et de méthodes, qui se développe de manière autonome.

Cette explicitation, si elle réussit, peut apporter un éclairage nouveau aux questions exposées plus haut. D'abord, il devient possible de penser conjointement les prétentions de la connaissance à la vérité *et* son ancrage dans les formes de vie ou dans l'expérience incarnée du monde. Une accusation courante qui touche toutes les positions évoquées plus haut est en effet celle de relativisme. Les lectures ethnologiques de l'idée de formes de vie chez Wittgenstein, par exemple, doivent admettre que le vrai dépend du contexte socio-culturel dans lequel des propositions sont énoncées. Ou encore, si on affirme, par exemple, que la technique se développe selon une logique autonome de celle de la science, on peut se demander quels en seraient les principes. Elle pourrait consister, comme le suggère Jan Sebestik à la fin de la citation donnée plus haut[123], en un processus d'adaptation continu et cumulatif de l'être humain à son milieu. Mais si tel est le cas, ne doit-on pas être prêt à admettre que ce développement n'a pas de fondement rationnel, qu'il suit seulement le hasard de nos utilités, et par conséquent, à abandonner toute prétention de la technique *à être une connaissance* ? De la même manière, si on affirme avec les partisans de la connaissance énactive que la situation est déterminée par l'agent dans l'action, et que nos concepts émergent de structures sensorimotrices récurrentes, n'est-on pas acculé à concéder, comme Varela, E. Thompson et Rosch le font bravement, qu'il n'y a pas de sens à parler d'un monde externe et objectif, existant indépendamment de ces situations de connaissance [124] ? Pour éviter l'ac-

123. Voir plus haut, § 15.
124. (Varela, E. Thompson et Rosch [1991] 2016, p. 172-178).

cusation de relativisme, toutes ces positions doivent donc également élaborer de nouveaux concepts de la connaissance et de la vérité, dont la logique de fondation serait tout autre que celle d'une correspondance entre la pensée et le réel.

C'est ici que la clarification du statut de validité ou de vérité des procédures peut entrer en jeu. Nous avons évoqué plus haut, avec approbation, l'interprétation logique du rôle de la notion de *forme de vie* dans la réflexion du dernier Wittgenstein. Cependant, limiter la portée logique de cette idée à celle d'un accord sans cesse renégocié entre les personnes sur la manière dont il faut voir les choses, sur ce qu'on peut faire, ce qu'on peut dire, etc., comme le fait notamment J. Floyd, pourrait manquer le contenu positif que portent en elle ces capsules de sens que sont les procédures. Quelle que soit sa relativité à des sous-jacents incertains, une procédure affirme avec certitude que, telles choses étant posées, et telles actions conduites, telles autres choses arrivent. Le fondement d'une telle assertabilité ne semble pas être inductif et empirique.

Ensuite, elle peut renouveler la compréhension du rapport entre les pôles théorique et pratique de la connaissance. La programmation et les procédures sont des lieux de circulation entre les sciences et les techniques, où les savoirs théoriques sont mobilisés pour l'action, mais où réciproquement des préceptes empiriques sont formalisés, reconfigurés, et rendus susceptibles de démonstration. Bien plus, comme nous l'avons évoqué, l'informatique nous permet de pointer plus précisément aussi bien le contenu procédural interne à tout concept, que les présupposés factuels sous-jacents à toute méthode, et ainsi de comprendre leur convertibilité essentielle l'un dans l'autre. C'est cela qui se trouve au fondement de la fluidité intrinsèque des échanges entre connaissance théorique et connaissance pratique. Elle se passe entièrement des chevilles conceptuelles par lesquelles on tente de l'expliquer habituellement, l'*application* et l'*abstraction*, auxquelles l'informatique assigne un rôle entièrement différent.

Ainsi, à travers ces débats concernant la fondation théorique ou pratique de la connaissance, s'exprime selon nous un seul problème central, qui concerne l'effectivité de la connaissance, et qu'il faut entendre en deux groupes de questions complémentaires.

D'abord, doit-on appeler *connaissance* cette compétence très spéciale qu'ont les personnes humaines de *concevoir* des solutions de leurs problèmes et d'agir conformément à ces conceptions ? Cette compétence (qu'on l'appelle pratique ou technique) ne semble avoir aucun rapport avec la connaissance au sens classique,

que la tradition caractérise par son orientation tendue vers l'objet « tel qu'il est », par le désintéressement et par la suspension de l'activité humaine. Aussi pourquoi les associe-t-on tout de même ?

Ensuite, comment se fait-il que la connaissance, au sens traditionnel, exhibe elle-même tous les caractères d'une pratique, susceptible de prescriptions et de techniques qui peuvent en augmenter l'effectivité – ce qu'atteste d'ailleurs la conception de la logique comme *organon* (outil) pour les Grecs, et comme « art de penser » pour les Modernes, ainsi que les « techniques cognitives » aujourd'hui ? Il semble que la connaissance, comme Athéna, naît d'emblée armée de significations procédurales qui lui permettent, non seulement d'être immédiatement effective dans le monde, mais également d'accroître progressivement son empire.

On peut reconnaître ici nos deux questions directrices[125] à propos de la connaissance des procédures. Il est logique en effet que la notion de *procédure* apparaisse comme essentielle au problème de l'effectivité de la connaissance ; et c'est aussi pour cette raison que l'informatique, qui nous en donne le concept *théorique*, mathématiquement analysable, joue un rôle de plus en plus central dans les débats mentionnés – qu'il s'agisse de technique, de logique, d'ancrage dans nos formes de vie, ou de processus cognitifs.

1.6 Situation dans le débat philosophique sur l'informatique

La problématique présentée dans cette introduction s'approfondit donc progressivement, à partir d'une première question sur l'effectivité et la polyvalence des ordinateurs dans notre monde humain, en direction d'une seconde concernant les liens entre programmation et connaissance, et enfin pointe vers une troisième encore plus élémentaire concernant l'effectivité de la connaissance en général et le sens qu'il y aurait à parler d'une vérité des méthodes.

D'un point de vue disciplinaire, notre travail s'inscrit donc aussi bien dans la philosophie de l'informatique que dans la philosophie de la connaissance. Il développe une interprétation de la nature de l'informatique afin d'élaborer une thèse plus générale concernant une certaine forme de connaissance. Mais cette

125. Voir plus haut, § 14.

interprétation est elle-même fondée sur une réflexion épistémique qui dépasse les bornes de l'informatique.

La philosophie de l'informatique est un champ interdisciplinaire par essence. Nous avons déjà évoqué les liens profonds de notre problème avec l'épistémologie des sciences, des mathématiques et des techniques, avec la logique et la philosophie du langage, ainsi qu'avec la science cognitive, bien qu'il ne puisse être assimilé aux problématiques propres de ces champs de recherche.

Nous avons également soigneusement démarqué notre approche des philosophies de l'information et des métaphysiques de l'esprit ou de la nature que celles-ci développent.

Dans cette section, nous achevons de situer notre travail dans le champ des questions qui animent aujourd'hui la philosophie de l'informatique. Nous commençons par décrire comment il pourrait contribuer, de manière indirecte, aux nombreuses questions éthiques et politiques que la « révolution numérique » suscite (§ 20). Nous évoquons ensuite son rapport aux questions consacrées à l'élucidation des notions même de programme et d'algorithme (§ 21), puis à celles ayant trait à la diversité des styles de programmation, à la spécification et à la vérification formelle, et enfin à l'implémentation (§ 22). Nous terminons cette section par l'annonce de notre plan (§ 23).

§ 20. La puissance de l'informatique

De très nombreuses réflexions sont en cours pour tenter de comprendre ce que les ordinateurs changent à notre monde humain, dans tous les domaines de la philosophie. Au niveau politique, il s'agit de comprendre la manière dont l'automatisation, l'Internet, les réseaux sociaux, l'intelligence artificielle, la blockchain*, modifient nos institutions, nos processus démocratiques, les rapports entre l'État, les individus, les organisations privées, et les communautés humaines, notamment en situation minoritaire. Au niveau éthique, il s'agit de comprendre comment ces mêmes technologies affectent nos comportements, le rapport à notre propre corps, nos relations avec autrui, nos cadres de décision et nos perspectives, individuelles et collectives.

Notre travail ne prétend contribuer qu'*indirectement* à ces débats majeurs, en mettant au jour les conditions de l'effectivité des programmes informatiques. Nous donnons ici une courte indication de la direction que cette contribution pourrait prendre, et que nous développerons un peu plus en conclusion de ce travail. Pour

ce faire, nous revenons sur le terme de *puissance de l'informatique* que nous avons mis de côté tout à l'heure, au profit de celui d'*effectivité*.

La puissance, appliqué à une technologie, est un terme ambigu. Il peut d'abord signifier la puissance complémentaire qu'il octroie à ses utilisateurs. On peut par exemple dire qu'un tableur* augmente la puissance de ses utilisateurs à faire des calculs. Plus généralement, on peut avancer par exemple que l'informatique accroît les possibilités qu'a chacun de s'informer, de communiquer, de travailler, de se divertir, etc. Mais dès qu'on entend ces assertions d'une manière globale, en disant que l'informatique augmente la puissance des personnes à faire ce qu'elles veulent, des débats complexes surgissent. On opposera en effet à ces « bienfaits » les conséquences catastrophiques d'Internet sur la lecture, la santé, la capacité de concentration, l'esprit critique, l'être-ensemble, etc.

On peut entendre également *puissance* relativement au programmeur. On peut par exemple dire qu'un informaticien a une certaine puissance sur un processus du monde réel par le biais du programme qu'il réalise – ce processus n'étant pas une réalité seulement informatique. Ainsi un logiciel d'édition de factures ne contrôle pas seulement les actions de l'ordinateur dédié à cette tâche, mais également, de manière indirecte, *celles du comptable qui les supervise*, puisqu'il lui impose ses normes et ses méthodes de traitement de factures. Nous reviendrons plus loin[126] sur le caractère bi-face de la prescription informatique, qui s'impose aussi bien à l'ordinateur qu'à l'utilisateur. En généralisant ce point, on peut comprendre que l'informatique permette à des entreprises de contrôler des domaines plus ou moins larges de l'économie et de la société – ainsi Amazon pour le commerce électronique, ou Facebook pour les réseaux sociaux. Il semble donc bien que les moyens informatiques confèrent une certaine puissance à des groupes humains, et qu'elle est un enjeu de pouvoir.

Or l'apparition de choses nouvelles peut induire des redistributions de pouvoir entre les personnes ou entre les groupes humains, sans pour autant en augmenter la quantité absolue, sans être *des progrès* à proprement parler. Ainsi, par exemple, l'informaticien Joseph Weizenbaum (1976, p. 31) avance l'idée que l'informatique, qui démultiplie les capacités de gestion et de contrôle, a servi à conforter l'ordre socio-économique existant plutôt qu'à être un facteur de progrès. Que l'ordinateur ait réussi à se rendre indispensable ne veut pas dire qu'il était nécessaire,

126. Au chapitre 3.

ni même utile :

> L'ordinateur devient une composante indispensable de toute structure une fois qu'il y est profondément intégré, et si emmêlé à ses diverses sous-structures vitales qu'il ne peut plus en être extrait sans lui porter un coup fatal. Cela est une tautologie. […] Il n'est pas vrai que le système bancaire américain ou les marchés boursiers et de matières premières, ou encore les grandes entreprises manufacturières se seraient effondrés si l'ordinateur n'était pas arrivé « juste à temps ». [Mais] il est vrai que la manière spécifique dont ces systèmes se sont réellement développés au cours des deux dernières décennies, et se développent encore, aurait été impossible sans l'ordinateur. [Et] il est vrai que si tous les ordinateurs disparaissaient soudainement, une grande partie du monde moderne industrialisé et militarisé serait plongé dans une grande confusion et peut-être un chaos total[127].

Il est difficile d'asseoir ou d'infirmer une telle hypothèse. Elle est l'objet d'un débat important chez les théoriciens des nouveaux media[128]. Même en se restreignant à un point de vue simplement économique, il n'est pas aisé de démontrer que l'informatique est un facteur déterminant de croissance, une difficulté qu'on appelle le paradoxe de Solow, selon la phrase célèbre de ce prix Nobel d'économie, énoncée en 1987 : « L'ère de l'ordinateur s'observe partout, sauf dans les statistiques de productivité[129] ».

Ici pointe un troisième sens possible de *puissance*, qui serait celle de l'informatique (ou plus généralement de la technologie) elle-même, se nourrissant de ses propres succès, et dont les intérêts humains particuliers, fussent-ils ceux du complexe militaire-industriel pointé par Weizenbaum, ne seraient que des moyens, des aliments pour sa propre expansion. Derrière les faits qui sont régulièrement dénoncés – le capitalisme de surveillance, l'insuffisance de la protection des données personnelles, les biais des algorithmes, la récusation du droit à l'oubli qu'imposent les réseaux sociaux et la blockchain*, des facteurs plus systémiques seraient à l'œuvre, qui dépassent les intérêts des gouvernements et des entreprises. La question n'est pas tant qu'ils exploitent l'informatique pour asseoir leur puissance, mais que cette forme de puissance soit elle-même si efficace qu'elle éclipse toutes les autres, y compris dans la conduite de la guerre.

On peut nier que ce troisième niveau d'analyse de la puissance de l'informatique soit pertinent ; attribuer une puissance à une technologie, c'est la personna-

127. (Weizenbaum 1976, p. 28).
128. Voir par exemple (Chun 2011 ; Edwards 1990 ; Frabetti 2014).
129. Voir (Triplett 1999) pour une analyse d'une première décennie de débats.

liser et la mythifier. Si une forme de puissance en chasse une autre, c'est qu'elle est plus *puissante*, justement. Cela est tautologique, et il n'y a là rien d'autre à voir dans la diffusion de l'informatique qu'une conséquence du darwinisme culturel qui régit le monde des idées et des techniques. Dennett (2017) défend exactement cette position, allant jusqu'à faire des artefacts numériques – ordinateurs et programmes – les formes auto-réplicantes les plus efficaces de l'évolution culturelle.

Heidegger prend, on le sait, une position inverse. Derrière l'impersonnalité et la neutralité de la cybernétique, se cache bien une volonté de puissance, qui anime le projet occidental d'instaurer à l'égard du monde et des étants qui le composent un rapport de contrôle et de maîtrise – d'où le *cyber-* (*gouvernement*) de *cybernétique*. Rendre les choses calculables est en effet nécessaire pour qu'elles soient planifiables et exploitables de manière sûre, *mises en sûreté*, selon l'expression de Heidegger, et c'est à cette finalité de calculabilité universelle que la science moderne serait occupée.

Ainsi, le caractère éminemment explicable des procédures informatiques, le « bon sens informatique » proposé par Gérard Berry pour dissiper nos « schémas mentaux inadaptés », tout ceci qui est présenté comme une évidence, est critiqué par Heidegger comme la conséquence du projet de la Métaphysique qui s'achève, de « sommer l'étant à rendre raison », c'est-à-dire de concevoir seulement comme étant ce qui peut être mesuré et calculé, afin de le mettre à notre disposition, sous notre pouvoir. Mais dans cette affaire, la personne humaine est tout autant dominée que l'étant qu'elle asservit ; elle subit ce principe de rationalité absolue qu'est la calculabilité :

> Dans l'histoire de l'humanité, la domination du puissant principe [de raison] est d'autant plus solidement assise que l'emprise de ce dernier sur toute représentation et tout comportement est plus générale, qu'elle est considérée davantage comme allant de soi et, partant, qu'elle passe plus inaperçue. Telle est aujourd'hui la situation.
>
> […] La technique moderne tend vers la plus grande perfection possible. Cette perfection consiste dans la complète calculabilité des objets. La calculabilité des objets présuppose la validité universelle du principe de raison. Enfin la domination, ainsi entendue, du principe caractérise l'être de l'époque moderne, de l'âge technique[130].

La calculabilité absolue dissout les objets en informations et en rapports d'informations, mais également la personne elle-même, de manière à rendre inopportune,

130. (Heidegger [1957] 2003, p. 254-255).

voire incongrue, sa posture méditative. Dans un tel « Dispositif », les sciences elles-mêmes sont redéfinies à partir de la cybernétique[131], et seule est possible l'admiration superficielle pour les progrès de la technique, à laquelle on s'en remet pour fixer l'avenir de l'être humain et résoudre ses problèmes.

La force de l'argument de Heidegger est de poser une question radicale : Y a-t-il, ainsi qu'il le prétend, une liaison intime entre *une certaine forme* de rationalité et la puissance de l'informatique ? Les débats éthiques et politiques qui ont lieu aujourd'hui sont profondément configurés par cette question, car ils orientent le type même de réponse qu'on pense devoir apporter aux problèmes de nos systèmes sociaux, économiques et politiques, entre réformisme et subversion.

En nous attachant à expliciter la dimension locale et interne de cette puissance, que nous appelons *effectivité*, c'est à cette question que nous espérons contribuer. Si le calcul est le destin de la pensée, il s'agit encore de savoir ce que ce terme recouvre, de quelles manières il s'infiltre dans nos manières de connaître jusqu'à nous sembler devenir une propriété des objets mêmes, et si ce mouvement a des limites visibles. Peut-on identifier connaissance et calcul, calcul et prédictibilité, prédictibilité et puissance ? qu'y a-t-il de spécifique à l'invention des ordinateurs qui permettrait une telle accélération de ce projet d'un contrôle universel ? et comment les ordinateurs – des machines à réaliser des « opérations mentales » sont-ils tout simplement possibles ?

Il s'agit là, en quelque sorte, de questions techniques, du point de vue de Heidegger. Mais elles doivent être traitées afin d'acquérir une appréciation plus exacte de la pertinence de sa thèse.

§ 21. La définition des programmes et des algorithmes

Notre questionnement est en relation assez directe avec de nombreuses questions abordées actuellement dans le domaine en plein essor de la philosophie de l'informatique. Celle-ci est, de manière majoritaire, ancrée dans la philosophie des sciences et des techniques : l'association d'histoire et philosophie de l'informatique[132] est une commission de l'Union internationale pour l'histoire et la philosophie des sciences et des techniques (IUHPST)[133]. Des réflexions sur la nature de l'informatique ont également lieu en études des media (*media studies*), puisque

131. (Heidegger 1990, p. 284).
132. History and Philosophy of Computing (HAPOC), www.hapoc.org.
133. www.iuhpst.org.

l'informatique y est considérée comme support des « nouveaux media[134] ». Elle y est donc étudiée quant à ses effets anthropologiques, culturels et sociétaux. Notre questionnement, par son contenu même, est cependant plus proche des problèmes abordés en philosophie des sciences et des techniques.

Nous mentionnons ici quelques problèmes actifs dans cette discipline qui, quoique distincts de notre propos, lui sont étroitement liés.

Un problème central de la philosophie de l'informatique est, naturellement, la définition des programmes et des algorithmes. Deux projets ANR sont actuellement en cours sur ces questions. Le premier, « PROGRAMme », dirigé par . De Mol[135], vise à appréhender la réalité des programmes dans sa pluralité : comme objets formels, comme notations (souvent textuelles), comme processus mécaniques, et comme éléments de systèmes techniques, sociaux et culturels. La thèse centrale de ce projet est que « Les programmes sont une nouveauté radicale, non au sens où ils n'auraient pas d'histoire, ou que rien de similaire n'aurait jamais été créé; mais plutôt au sens où ils ont abouti à une nouvelle situation (que l'on pourrait appeler la société numérique) qui requiert d'être repensée à neuf[136] ». Le second projet, « Geometry of Algorithms », dirigé par A.Ñaibo[137], vise à « donn[er] une représentation mathématique précise aux algorithmes », notamment par l'utilisation d'outils géométriques, ce qui « ouvrira ainsi la voie à l'élaboration d'une ontologie propre à l'informatique, et à considérer celle-ci comme une théorie mathématisée, au même titre que la physique. »

Ces deux projets nourrissent notre réflexion[138]. Nous abordons cependant l'informatique sous un angle différent et complémentaire. Nous ne nous demandons pas en effet *ce qu'est un programme* ou un algorithme, mais *ce qu'est programmer*. Cette inflexion est importante et exige un commentaire.

Lorsqu'on centre la réflexion sur un objet, comme un programme, on se demande naturellement de ce qu'il est, de quoi il est composé, et quelles sont ses relations avec d'autres objets. Cette optique ne peut convenir à nos deux questions, qui concernent la connaissance impliquée dans la programmation ainsi que son effectivité. En effet, la question de la connaissance devient celle de la « correspondance » entre le programme et l'objet qu'il dénote (dont il faut décider s'il

134. (Manovich 2002).
135. programme.hypotheses.org .
136. Collectif PROGRAMme, *What is computer program ?*, à paraître.
137. anr.fr/Project-ANR-20-CE27-0004.
138. Voir notamment, plus loin, § 39 et § 41 et chapitre 9.

s'agit d'un objet formel, d'un objet sémantique, ou d'un objet réel dans le monde), et celle de l'effectivité devient celle de la « production » par le programme (ou par l'utilisateur du programme, ou par la machine) d'effets (dont il faut décider s'il s'agit de modifications d'états de l'ordinateur, qu'il faut également décrire, ou de l'environnement plus large du programme, qui inclut des utilisateurs, des objets physiques, d'autres ordinateurs, etc.). Cette formulation de nos questions ne nous semble pas pertinente, car elle introduit en leur sein de nombreux nouveaux problèmes qui ne s'y trouvent pas de prime abord.

Notre parti-pris est de partir de la *pratique* même de la programmation, comme activité de connaissance et de préparation de machines à l'exécution de tâches qui leur sont confiées. Ce parti-pris nous rapproche, d'une certaine manière, de la philosophie dite des pratiques mathématiques[139], qui s'attache aux processus mêmes par lesquels les mathématiciens déterminent leurs concepts et leurs questions, élaborent leurs preuves, et les valident. Si on considère un programme comme un objet mathématique, notre démarche a les mêmes justifications que cette approche philosophique. Elle est également justifiée en ce que la programmation est une technique et une pratique professionnelle. Dans cette seconde perspective, nous adoptons ici résolument le point de vue du producteur plutôt que celle de l'utilisateur, étant donné la nature de nos questions.

Ce point est d'autant plus pertinent concernant la programmation qu'il est de plus en plus rare aujourd'hui de rencontrer des programmes qui ressemblent à des « produits finis », ou encore à des artefacts manufacturés. La vision artefactuelle des programmes fut adoptée aux débuts de l'informatique personnelle (à partir de la fin des années 1970) par les éditeurs de logiciel qui cherchaient à commercialiser des programmes standardisés (jeux et bureautique principalement) et auxquels il fallait donner l'apparence de produits matériels qu'on pouvait acheter dans des commerces, pour lesquels le consommateur trouverait légitime de payer un prix. Cependant, un programme n'a que très rarement le « comportement » d'un produit manufacturé fini. Cette thèse est importante et nous devons nous attarder à la justifier :

1. D'un point de vue économique, il est courant pour les organisations de rémunérer l'éditeur en *licences d'usage* annuelles et souvent proportionnelles au nombre d'utilisateurs. Ce modèle de paiement est pertinent pour elles car il leur permet de proportionner leurs paiements à leur intensité d'usage, et

139. Voir (Mancosu 2008) pour une introduction.

d'accéder à un service après-vente ainsi qu'à des mises à jour techniques. Le développement du logiciel libre, au début des années 2000, a considérablement orienté en ce sens le modèle d'affaires de l'industrie du logiciel.

2. Par ailleurs, les mises à jour fonctionnelles sont à présent très régulières, et il n'y a plus de sens pour les clients à attendre trois à quatre ans avant d'acheter une nouvelle version de leur logiciel. Aussi le modèle majoritaire aujourd'hui, y compris pour les consommateurs, est-il de « s'abonner » au logiciel comme à un service, qui garantit l'accès à la dernière version disponible.
3. La représentation du logiciel comme d'un « produit » a achevé sa transmutation en celle d'un service avec la généralisation de l'informatique en nuages*, où l'accès aux fonctionnalités requises se fait à distance, l'ordinateur des utilisateurs n'hébergeant plus de manière permanente le « code » du programme. Le nom de ce modèle de distribution du logiciel s'appelle, de manière significative, « logiciel en tant que service » (*Software as a Service*).
4. Toutes les remarques ci-dessus s'appliquent aux logiciels standardisés. Ceux-ci ne constituent en général qu'une faible partie du système d'information* des grandes organisations, qui disposent d'une équipe informatique développant, supervisant et maintenant des logiciels « maison ». Ces programmes n'ont qu'une seule instance d'exécution, qui n'est pratiquement jamais mise à l'arrêt, et qui est constamment surveillée et reconfigurée. Les mises à jour sont fréquentes. Le « service » est ici continu est entièrement personnalisé, si on peut s'exprimer ainsi.

Ces remarques ont une valeur suggestive. Elles aident à nous détacher d'une vision naturellement « artefactuelle » des programmes, et à les considérer plutôt comme des services rendus par des « programmeurs » à des « utilisateurs » (qui peuvent être la même personne) par le biais de machines, ou encore mieux, comme des méthodes instanciées par le biais de machines. Les représentations de *service* et de *méthode* sont ici utiles en ce qu'on perçoit mieux qu'ils puissent être *eux-mêmes* des gestes opératoires, des processus de modification du réel, sans cesse personnalisés et adaptés à la situation requise, *par un programmeur* dont on ne peut vraiment faire l'abstraction afin de traiter nos questions. Il s'agit là d'un changement de perspective sur la nature de la programmation, notamment porté par Naur (1985) et C. Floyd (1993), qui refusent de la réduire à une activité logique ou formelle, et insistent sur sa dimension participative :

> Le système référent [la réalité dans laquelle s'insère le logiciel et qu'il vise à accompagner] est [une] imbrication de processus de travail, d'apprentissage et de communication, [...] il est dynamique et évolutif dans le temps. Le système logiciel va contribuer à faire évoluer l'environnement, qui va à son tour réagir sur le système référent de manière imprévisible. Le développeur du logiciel devient partie intégrante du système référent ; les processus de développement et d'utilisation sont considérés comme s'influençant mutuellement[140].

Ces remarques permettent également de préciser quelles activités recouvre la programmation dans notre réflexion. Nous ne désignons pas seulement par ce terme l'activité de *codage*, c'est-à-dire l'élaboration et la gestion de notations procédurales susceptibles d'être évaluées par un ordinateur. D'abord, la programmation recouvre l'ensemble des étapes du développement d'un logiciel, dont le codage n'est qu'une faible partie. Comme on le verra, le développement, qui s'organise en projet, comprend également la description des besoins, la spécification, les tests, la validation et l'implémentation ou mise en production. Mais là ne s'arrête pas le périmètre de ce que nous appelons programmation. En effet, au sein d'une équipe informatique, les frontières entre développement et « opérations » est poreuse. Ce dernier terme désigne les activités de supervision de l'exécution du logiciel, de maintenance du matériel, de configuration de ses paramètres, de gestion de la sécurité, etc. Toutes ces activités influencent le comportement des programmes et des machines et doivent donc être comptées pour de la programmation, même lorsqu'elles ne « codent » pas. La diffusion du modèle Devops* d'organisation des équipes informatiques, où celles-ci sont responsables conjointement du développement et des opérations d'une partie du système d'information, montre qu'il n'y a pas grand sens à limiter strictement l'activité de programmation à celle de développement. Plus généralement encore, tous les métiers de conception et de gestion de systèmes informatiques entrent dans le périmètre de ce que nous appelons programmation dans le cadre de notre travail. Ainsi, lorsque Peter Denning dénonce l'identification courante de l'informatique avec la programmation, en évoquant tous les autres métiers qu'elle comporte : « architectes systèmes, ingénieurs réseaux, de systèmes d'exploitation, de bases de données, graphistes, architectes et concepteurs de systèmes logiciels, experts en sécurité, en simulation, en réalité virtuelle, en calcul intensif, roboticiens[141] » – cette critique ne nous concerne pas, car nous prenons la programmation en un sens bien plus large que l'activité de codage

140. (C. Floyd 1993, p. 246).
141. (Denning 2004, p. 16).

ou de développement logiciel.

C'est la réalité dynamique de ces activités que notre travail doit permettre de mieux appréhender. En quoi sont-elles des activités cognitives ? et comment peuvent-elles, à l'aide de machines, être effectives dans leur environnement ? Afin de répondre à ces questions, il ne faut pas prendre pour point de départ les programmes ou les algorithmes, mais bien la programmation elle-même.

§ 22. Autres problèmes de la philosophie de l'informatique

Un second problème, étroitement lié au précédent, concerne la multiplicité des formes de programmation. Il existe sans doute plus de 9000 langages aujourd'hui, et au moins 700 qui sont actifs. Au-delà de cette pléthore sans doute anecdotique, plusieurs paradigmes de programmation[*142] sont distinctement identifiés, comme les paradigmes fonctionnel, impératif, logique, sous contraintes, orienté-objet, concurrent, etc. Chacun de ces styles donne une idée assez différente de *ce qu'est* une procédure ou un calcul, de telle sorte qu'il modifie la manière même dont chaque programmeur réfléchit aux problèmes qu'il a à résoudre, ainsi que l'illustre Backus (1978) lorsqu'il promeut l'idée du style fonctionnel de programmation. Selon lui, la domination du style impératif découle naturellement de l'architecture dominante de machine programmable, « l'ordinateur de Von Neumann », du nom du célèbre mathématicien hongrois qui en posa les bases[143], mais elle n'est pas la seule possible. Une telle idée, que seraient possibles des formes de programmation radicalement différentes de celles que nous connaissons, si nous disposions de concepts radicalement différents de ce qu'est un ordinateur, est aujourd'hui plausible. Des processeurs répliquant l'architecture des réseaux de neurones sont en cours de développement, et il y aura peut-être bientôt des ordinateurs quantiques et biologiques. Chacune de ces typologies de machines se programme selon des logiques entièrement différentes. En même temps, tous ces styles de programmation et toutes ces architectures couvrent le même domaine de fonctions calculables selon la thèse de Church-Turing[*] (avec un doute quant aux ordinateurs quantiques).

Cette multiplicité apparemment indéfinie de « manières de penser » les procédures effectives, mais qui finalement « reviennent au même » d'un certain point de vue mathématique, est source de perplexité. D'abord, cette tension rend difficile la

142. Sur la notion de paradigme de programmation, voir l'article historique de Priestley (2017).
143. (von Neumann [1945] 1993).

caractérisation d'un critère *d'identité* pour les programmes et les algorithmes, ce qui rejaillit sur le problème de leur définition mentionné plus haut. Surtout, il est difficile de comprendre d'où vient cette multiplicité, ce que voudrait dire ici *manières de penser*, ou encore *style*, dans un domaine où l'on cherche à débarrasser la réflexion de tout élément subjectif et arbitraire. Cette difficulté est bien entendu un paramètre important de notre réflexion concernant la programmation comme activité de connaissance.

Un autre problème important qui occupe la philosophie de l'informatique concerne la manière dont il faut comprendre le processus de « traduction » d'un problème réel en spécifications formelles puis en code informatique. Dans la plupart des cas – sauf lorsque le problème est lui-même déjà exprimé formellement – il semble que cette traduction doive passer par un travail de modélisation, et on peut alors se demander si ce travail est similaire à celui qui a lieu en sciences. Une approche courante est d'opposer le modèle scientifique comme descriptif, et celui de l'informaticien comme prescriptif (l'image est ici celle du modèle de l'architecte, qui dit à quoi *devra* ressembler la maison) mais une telle opposition est, comme on le verra, pleine de difficultés. Une seconde question liée à ce problème de traduction concerne le rôle qu'y peuvent jouer les méthodes formelles*. Initialement inventées pour vérifier rigoureusement l'adéquation du code informatique à sa spécification, elles sont devenues aujourd'hui des langages à part entière qui permettent d'expliciter directement le modèle du problème qu'il s'agit de résoudre. Ces questions nous intéressent particulièrement, car elles sont au cœur de la dimension cognitive de la programmation. Cependant, comme nous l'avons dit, nous ne les aborderons pas comme des questions de *traduction* – puisqu'il existe des cas pratiques où le programmeur code directement la solution d'un problème énoncé informellement.

Le quatrième problème a déjà été mentionné[144] : il s'agit tout simplement de ce que nous avons appelé le problème de l'effectivité des ordinateurs. Comment est-il possible que des significations soient transformées en phénomènes matériels sans l'intervention d'un agent humain qui les interprète ? En philosophie de l'informatique, ce problème est connu comme le *problème de l'implémentation*[145], et il est souvent traité comme un problème de traduction ou de correspondance. Il

144. Voir plus haut, § 3.
145. (Chalmers 2012).

a été initialement formulé sous la forme d'un paradoxe que tout objet physique obéissant à une loi mathématique du mouvement pouvait être dit « calculer » cette loi[146]. Cela a requis des philosophes qu'ils caractérisent mieux la relation *spéciale* qui devait tenir entre significations et phénomènes matériels afin qu'on puisse proprement dire qu'*un système calcule*. Ce problème est lié à une autre question, de savoir si on pouvait parler de *calcul* pour décrire le fonctionnement de systèmes non programmés, notamment naturels, comme les processus neuronaux ou le codage - décodage de l'ADN. Il s'agit de la question du *calcul physique* (*physical computation*), ainsi nommée car elle vise à donner une définition physique de la notion de calcul, dont l'exécution de programme ne serait qu'une modalité parmi d'autres.

Nous mentionnons un autre problème important, mais que nous n'aborderons pas dans notre travail. Il concerne la validité de la thèse de Church-Turing*, et notamment la possibilité de dispositifs matériels calculant *plus* de fonctions qu'une machine de Turing, ce qu'on appelle *hyper-computation*. Nous n'entrons pas dans ce débat complexe, qui a trait à la conception de l'infini. Une question de philosophie de l'esprit – qui donne une grande partie de son intérêt au débat – est de savoir si l'esprit humain est capable d'hyper-computation. Cela établirait que les machines « ne peuvent penser comme nous[147] ».

Il est enfin important de préciser que notre travail *ne porte pas* sur l'apprentissage machine*, la classe dominante d'algorithmes utilisés aujourd'hui pour résoudre les problèmes dits d'« intelligence artificielle » (IA), dont nous avons vu un exemple plus haut avec la reconnaissance faciale. Notre travail ne peut élucider les enjeux de connaissance propres à ces algorithmes, car leur particularité essentielle est leur nature statistique, et c'est donc une épistémologie de la mesure, de l'induction, de la probabilité et de la statistique qui est requise afin d'élucider les problèmes qu'ils suscitent, et notamment l'essentiel des phénomènes d'opacité, c'est-à-dire d'inaccessibilité structurelle à la compréhension humaine. Nous présentons une esquisse de réflexion à ce sujet en conclusion de ce travail[148].

146. Voir (Piccinini 2015, p. 16-25) pour l'exposition de la difficulté.
147. (Bringsjord et Zenzen 2003).
148. Voir plus loin, section 13.2.

§ 23. Approche

Notre travail procède en cinq parties. Dans la partie **I Qu'est-ce que programmer ?**, nous montrons la difficulté à cerner la programmation comme phénomène technique, socio-économique ou psychologique ; elle est protéiforme et se transforme sans cesse (chapitre 2). Nous la caractérisons néanmoins comme un processus épistémique complexe qui vise à insérer une machine dans une situation afin d'y résoudre un problème (chapitre 3).

Cette caractérisation nous commande d'élucider ce qu'est une résolution de problème et ce qu'est une machine. Ce travail occupe les trois parties centrales de ce travail. Dans la partie **II La forme générale de la raison pratique ?**, nous prenons pour fil directeur la proposition attractive de Herbert Simon, que ce serait au contraire dans la programmation que la résolution de problème aurait atteint sa forme exacte, et qu'il faudrait donc comprendre la seconde à partir de la première. Les apports de cette approche sont nombreux, mais en postulant que toute investigation procède de manière réglée, elle ne fait pas droit à la fluidité et à l'ambiguïté des délibérations pratiques. Nous nous accordons plutôt avec les philosophes de la pratique pour montrer que les règles techniques émergent des pratiques, plutôt que le contraire (chapitre 4). Les procédures informatiques, quant à elles, doivent être considérées comme relevant d'une troisième strate réflexive. Comme les procédures industrielles, elles sont des reconstructions complètes des processus de délibération humains, qu'elles expriment dans le cadre de systèmes de règles préétablis. Loin d'être de simples précisions ou reformulations des intuitions pratiques ou des règles techniques, leur logique est souvent en opposition frontale avec elles (chapitre 5).

La partie **III Théorie et pratique** vise à étudier, dans l'histoire de la philosophie, la conception de la *possibilité* de cette troisième strate de savoir pratique, *i.e.* de résolution *systématique* de problème (ce qu'est la *systématicité* restant à élucider). Si Aristote montre la possibilité d'élaborer des règles techniques, ainsi que celle de conduire des investigations scientifiques systématiques, il reste dubitatif sur la possibilité de systèmes techniques, du fait de leur ouverture à la contingence des circonstances humaines (chapitre 6). On voit cependant, au Temps modernes, se constituer un vaste mouvement de rationalisation des pratiques, les réductions en art, qui butent cependant sur la forme de systématicité qu'on pourrait leur conférer, confirmant ainsi le scepticisme aristotélicien. C'est dans la mécanique que se fait la percée décisive : la machine, que les modernes conçoivent comme

une construction géométrique, est un lieu d'exactitude qui donne d'ailleurs son modèle à la nouvelle science de la nature (chapitre 7). Kant généralise ce résultat, en déclarant que les *règles* par lesquelles nous formons nos jugements sur l'expérience sont celles-là même qui peuvent guider l'action technique. Il n' y a donc plus d'interdiction de droit à développer des techniques systématiques. Cependant, nous ne savons toujours pas ce que seraient de tels systèmes de règles qui portent, non pas sur les jugements de faits, mais sur les transformations de tels jugements dans l'investigation (chapitre 8).

La partie **IV la résolution de problème** achève d'élucider la possibilité d'une résolution systématique de problème, en prenant d'abord pour fil conducteur les mathématiques. La théorie des types de Per Martin-Löf, qui permet de concevoir la co-appartenance du jugement et de sa preuve, ou encore de la construction et de sa procédure de construction, nous amène à dégager trois principes qui président à l'approche systématique des problèmes (chapitre 9). Nous achevons cette analyse en plaçant tous ces résultats dans le cadre de la *théorie de l'investigation* de John Dewey, qui permet d'articuler les trois strates de savoir pratique – à savoir la délibération « intuitive », la règle technique et la procédure industrielle ou informatique – dans leur continuité mais également leur différence radicale. Le point essentiel est qu'une investigation procède *toujours* par la reconstruction itérative de la représentation du problème (chapitre 10).

Dans la partie **V Machines et systèmes**, la programmation nous apparaît donc comme la reconstruction procédurale d'une investigation réussie, dans un système de règles prédéfini. Cela nous permet d'achever l'élucidation du concept de *machine* et de montrer la possibilité de machines universelles capables d'exécuter toute procédure établie dans un système de règles quelconque. Cette possibilité bouleverse l'investigation elle-même, en lui permettant de manipuler des procédures délibératives comme des objets quelconques (chapitre 11). Mais le fait essentiel que la machine est *interactive* conduit à un bouleversement plus grand encore ; en rendant les règles directement effectives dans les situations humaines, ce sont tous nos systèmes institutionnels et techniques qui deviennent dynamiques, et par là, incertains (chapitre 12).

Notre **conclusion** reprend les questions qui ont émergé au cours de ce travail. Deux d'entre elles sont l'objet d'une analyse complémentaire, l'une liée à l'apprentissage machine statistique, et l'autre à la nature de l'effectivité des procédures. Enfin, nous évoquons l'apport possible de notre travail aux questions éthiques et

politiques engendrées par les bouleversements du numérique.

Première partie

Qu'est-ce que programmer ?

Chapitre 2

Premières approches

§ 24. Introduction

Nous cherchons dans ce chapitre à circonscrire la programmation par une série d'observations, de manière à pouvoir en donner une première caractérisation ou définition. L'ambition d'une telle « leçon de choses » est immense, étant donné, on va le voir, les formes très diverses que prend cette activité humaine – qu'il s'agisse de ses formes techniques, sociales ou économiques. Or c'est justement ce caractère protéiforme de la programmation qu'il est bien important de mettre en évidence au début de cette réflexion, afin qu'une vision unilatérale – *« Il s'agit d'une activité technique ! »*, ou *« Il s'agit d'une profession comme une autre ! »* ou encore *« Ce sont des calculs mathématiques ! »* et *« C'est de la modélisation ! »* – ne masque les multiples facettes et arrière-plans qu'elle recèle, et dont il nous faut chercher le point de convergence.

La longueur de ce chapitre s'explique donc par la nécessité de constituer un référentiel d'observations partagées avec la lectrice, dont nous ne supposons pas qu'elle a une quelconque connaissance en informatique, sinon celle d'une utilisatrice courante de notre époque. Par conséquent, nous présentons chaque exemple de programme avec assez d'explications pour qu'une lectrice parcourant le code associé *avec attention* puisse en saisir la trame, quand bien même elle ne connaîtrait pas les langages informatiques associés. Ces développements pourront par contre paraître fastidieux à la lectrice experte, à laquelle il est recommandé d'aller directement à leurs conclusions. Programmer est une *activité* et, à ce titre, la lecture passive de sa translittération textuelle est peu gratifiante.

L'effort d'expliquer en langue française le contenu de plusieurs petits fragments

de code a une autre vertu pour ce travail. En effet, cet exercice permet de mettre en évidence la réflexion sous-jacente qui guide le programmeur dans ses décisions, et ainsi de montrer que, en-deçà de l'expression aride du code informatique, coule le flot souterrain du sens commun dont nous avons tous l'expérience.

Cet effort nous a amené à choisir des exemples très simples de programmation, à la fois pour mettre notre propos à la portée de l'audience la plus large possible, mais également pour réduire l'ampleur des explications nécessaires qui sont déjà copieuses même pour des petits programmes. Nous nous excusons par avance auprès des informaticiennes de la candeur conséquente de nos exemples.

La difficulté de définir ou caractériser la programmation ne tient pas seulement à la diversité de ses formes. Celle-ci découle d'un problème plus large encore : la programmation se transforme sans cesse. Elle est une des activités centrales de ce qu'on appelle les *nouvelles technologies* et ce qui apparaît comme sa meilleure pratique un jour peut paraître dépassé un peu plus tard. Comme le dit une des citations rapportées plus loin[1], le développement logiciel connaît des modes dont la fréquence semble plus caractéristique de l'industrie de l'habillement que de l'ingénierie. Au-delà de ces transformations rapides qui fragmentent la programmation en pratiques diverses et difficiles à résumer dans l'unité d'une définition, un mouvement plus profond a lieu, qui tient à sa démocratisation ou plutôt à sa *dissolution* dans des usages qui reprennent ses fonctions et ses pouvoirs en s'évitant cependant l'écriture de code. Ces usages – comme par exemple celui du tableur* que nous illustrons dans ce chapitre – ont une frontière floue avec la programmation, et ce flou devient intrigant lorsqu'on le rencontre dans des logiciels ou des langages de programmation au service d'autres activités humaines comme le droit, les mathématiques, l'architecture, la stratégie militaire. Ce n'est plus en effet seulement l'utilisation de l'ordinateur qu'il devient difficile de distinguer de la programmation, mais bien l'activité elle-même – l'application d'une loi, la démonstration d'un théorème, la conception d'un bâtiment – tant la programmation semble avoir la capacité à s'infiltrer dans la logique même des pratiques qu'elle soutient.

Notre première section vise à décrire la programmation « de l'extérieur » comme une activité économique et sociale importante qui occupe aujourd'hui quelques 27 millions de personnes dans le monde. L'analyse des statistiques met en évidence la difficulté de circonscrire leurs pratiques. D'une part, il est difficile de distinguer le professionnel de l'étudiant et de l'amateur qui, dans

1. Voir plus loin, § 34.

l'open-source*, peuvent être réunis tous trois dans un même projet, une possibilité qui caractérise de manière unique, pensons-nous, l'industrie du logiciel. D'autre part, il est difficile de distinguer le développeur attitré d'autres informaticiens et surtout de toutes les autres personnes – ingénieurs, scientifiques, cadres en entreprise – qui programment dans le cadre de leur activité professionnelle sans être informaticiens. Dans les sections 2.2 et 2.3 nous tentons de caractériser la programmation de manière *axiologique* grâce à l'observation de ses *meilleures pratiques*, telles qu'elles apparaissent dans l'ingénierie logicielle professionnelle, en la contrastant avec la programmation élémentaire. Dans les sections 2.4 à 2.6 nous tentons de la caractériser de manière *analytique* en cherchant une définition commune à toutes ses formes, y compris les plus diverses que nous pouvons observer[2]. Ces deux tentatives, bien que riches d'enseignements, se révèlent insuffisantes pour notre projet. Elles nous amènent à la conclusion que la programmation est d'abord un procès épistémique particulier, que nous examinons au chapitre suivant.

2.1 La difficulté à cerner la programmation

§ 25. La fin de la programmation

Un des articles éditoriaux du numéro paru en janvier 2023 de la revue phare de l'ACM, la principale association d'informaticiens, s'intitule *The End of Programming*, « La fin de la programmation »[3]. On ne saurait choisir de plus mauvais moment, semble-t-il, pour présenter une thèse de philosophie concernant ce sujet : comment en effet convaincre une plus large audience de l'importance d'une notion technique, lorsque les techniciens eux-mêmes semblent prêts à l'abandonner ?

Pour Welsh, l'auteur de cet article, l'expression *écrire un programme* n'aura un jour plus cours, car la manière même par laquelle on instruira les ordinateurs à résoudre nos problèmes va fondamentalement se transformer. Welsh s'appuie sur l'idée récente que des algorithmes d'apprentissage constitués de milliards de paramètres déjà calibrés sur des ensembles de données aussi vastes que l'Internet pourraient apprendre à réaliser de nouvelles tâches très rapidement, c'est-à-dire grâce

2. Sur la définition analytique, voir (Robinson 1963, p. 174), qui reprend Moore, et (Gupta 2015).
3. (Welsh 2023).

à quelques exemples (*apprentissage en quelques coups*, *few shot learning*). Cette idée provient du domaine du traitement automatique du langage naturel*, où les très grands modèles comme ChatGPT parviennent à générer des textes complexes en quelques interactions avec le requérant[4], mais également des programmes informatiques : en février 2023, Co-Pilot, le service d'aide à l'écriture de programmes sur la plateforme populaire de développement GitHub voyait près d'une de ses suggestions sur deux acceptée par les développeurs[5]. Welsh conclut que le futur de la programmation consistera à simplement décrire nos problèmes à des systèmes déjà entraînés grâce à des exemples :

> Les ingénieurs du futur pourront générer, en quelques frappes sur le clavier, l'instance d'un modèle à quatre quintillions de paramètres, encodant déjà toute l'étendue des connaissances humaines (et plus encore), prêt à se voir confier n'importe quelle tâche. L'essentiel du travail intellectuel requis pour diriger la machine à faire ce que l'on veut consistera à trouver les bons exemples, les bonnes données d'apprentissage et les bonnes façons d'évaluer le processus d'apprentissage. […] Dans cette nouvelle informatique – si nous l'appelons encore informatique – les machines seront si puissantes et sauront déjà faire tant de choses que le domaine ressemblera moins à de l'ingénierie qu'à de l'éducation, puisqu'il s'agira de savoir comment éduquer au mieux la machine […] Contrairement aux enfants (humains), cependant, ces systèmes d'intelligence artificielle piloteront nos avions, exploiteront nos réseaux électriques et gouverneront peut-être même des pays entiers[6].

Le style enflammé de ce texte, caractéristique des techno-visionnaires de la Silicon Valley, peut certainement faire sourire. Cependant il ne s'agit pas ici de contester la possibilité de la vision proposée. Le texte de Welsh nous intéresse parce qu'il met en évidence la difficulté de cerner la nature de ce qu'est la programmation.

À l'appui de sa thèse, Welsh utilise un argument *par le précédent*. Il part du constat que la programmation aujourd'hui n'a plus grand chose à voir avec ce qu'elle était aux commencements de l'informatique :

> Les pionniers […] croyaient fermement que tous les futurs informaticiens devraient avoir une compréhension approfondie des semi-conducteurs, de l'arithmétique binaire et de la conception de microprocesseurs pour comprendre ce qu'est un logiciel. Aujourd'hui, je suis prêt à parier que 99 % des personnes qui programment (*who write software*) n'ont presque aucune idée du fonctionnement réel d'un processeur[7].

4. (T. B. Brown et al. 2020).
5. (Dohmke 2023).
6. (Welsh 2023, p. 2).
7. (ibid., p. 1).

De manière similaire, argumente Welsh, les experts de l'apprentissage machine* aujourd'hui parviennent à réaliser des applications sans savoir programmer, au sens habituel qu'on donne à ce mot aujourd'hui. Mais il ne faut pas voir là une dégradation des compétences des informaticiens : Welsh cite à l'appui une méthode d'apprentissage profond* exposée dans un récent journal académique : son niveau de technicité la rend réciproquement incompréhensible aux programmeurs classiques et, se rangeant lui-même dans cette dernière catégorie, il se qualifie de « dinosaure ».

L'argument de Welsh met en évidence la difficulté à cerner notre sujet. Car doit-on assimiler la programmation à ses formes actuellement dominantes ? Mais dans ce cas, que sont l'arithmétique binaire et l'apprentissage machine* pour qu'on puisse dire que la programmation *supplante* la première et qu'elle est *supplantée* par la seconde ? Pour désigner la fonction commune à laquelle toutes trois répondent successivement, Welsh doit utiliser des périphrases comme « diriger la machine à faire ce que l'on veut » (*getting the machine to do what one wants*) dans la première citation donnée plus haut.

On a là un problème de vocabulaire, car en un autre sens plus large, c'est exactement de cela que la programmation s'occupe. L'apprentissage machine* n'en serait donc qu'une forme nouvelle, et non le remplacement. À l'appui de cette acception, on trouve tôt l'idée que la programmation est protéiforme et qu'elle se réinvente sans cesse, notamment dans cet épigramme du célèbre informaticien Alan Perlis à propos des langages de programmation :

> Un langage qui n'affecte pas votre façon de penser à ce qu'est programmer ne vaut pas la peine d'être connu.
> *A language that doesn't affect the way you think about programming, is not worth knowing* [8].

On pourrait dire de l'aphorisme de Perlis qu'il est si vrai qu'il montre ses propres limites, puisque l'apprentissage machine* remet en cause jusqu'à l'idée même de *langage* de programmation. Or c'est bien cela qui intéresse notre réflexion, de comprendre *par quelles méthodes* on parvient à « diriger les machines à faire ce que l'on veut », c'est-à-dire à les rendre effectives dans le monde.

Cependant, à l'appui de l'argument de Welsh, il est vrai qu'on n'utilise plus le verbe *programmer* pour désigner l'apprentissage machine*. On dira plutôt qu'on *entraîne*, qu'on *construit* ou qu'on *créée* un modèle d'apprentissage, mais beau-

8. (Perlis 1982).

coup moins souvent qu'on le programme. Et c'est là qu'on peut parler, en un sens plus général, de *fin de la programmation*, en ce que cette notion deviendrait progressivement obsolète, non pas principalement du fait des progrès de l'intelligence artificielle, mais d'abord parce qu'elle deviendrait inutile et inadéquate à désigner les diverses modalités par lesquelles nous « dirigeons les machines à faire ce que l'on veut ».

Cette obsolescence se manifeste d'abord dans la baisse relative de la fréquence du terme *programme* et ses dérivés (*programmer, programmation*, etc.), telle que peut la mesurer Google Ngrams dans le corpus d'ouvrages anglo-saxons numérisés par cette entreprise. Malgré les défauts et les biais de cet outil de mesure[9], il indique, dans ce cas précis, une tendance dont l'ampleur semble significative. Après s'être accrue rapidement des années 1950 à 1980, dans le contexte du développement de l'informatique. cette fréquence est maximale en 1980, lorsqu'apparaît l'informatique personnelle. Elle entame cependant après cela une dégringolade à la même vitesse jusqu'à aujourd'hui, où le terme atteint une fréquence similaire à celle de la période pré-informatique. On observe un même mouvement dans le corpus français, où le déclin du terme s'opère plutôt entre les années 2000 et 2015 et est d'une amplitude moindre.

Cette décroissance peut avoir de multiples causes. La programmation devient un phénomène de société au début des années 1980, avec l'avènement de l'informatique personnelle ; depuis elle a perdu une partie de son prestige. Le grand public est devenu familier et surtout consommateur averti des applications et des logiciels informatiques. Cela a émoussé son émerveillement face aux prouesses du programmeur et l'a au contraire rendu sensible à ses manquements – bogues, ergonomie défaillante, obsolescence rapide. Aussi la programmation s'est-elle vue reléguée à l'« arrière-boutique » de l'informatique.

Cependant il nous semble qu'au-delà de cette banalisation, une autre forme d'invisibilisation, bien plus importante, est également en cours. L'informatique s'est, dans de nombreux domaines, simplifiée et démocratisée, de telle sorte que la programmation semble être devenue inutile. Par exemple, on peut aujourd'hui réaliser un site web sans écrire de code (grâce à des logiciels de publication comme WordPress), mais également des chaînes complexes de calcul (grâce à des tableurs* comme Excel). De l'autre côté du spectre, de plus en plus de personnes programment dans le cadre de leur activité professionnelle – scientifiques et

9. À ce sujet, voir la mise au point de (Younes et Reips 2019).

ingénieurs notamment, mais ce n'est pas ainsi qu' un mathématicien ou qu'un ingénieur décriraient leurs activités sur Matlab ou Revit. Le premier dirait par exemple qu'il visualise des comportements de fonctions, et l'autre, qu'il explore une piste concernant la structure d'un bâtiment.

Haigh et Ceruzzi, concluant leur histoire de l'informatique sur l'exemple du véhicule électrique Tesla, montrent comment le logiciel a pris le contrôle de tous les sous-systèmes constitutifs de cette voiture (moteur, freinage, suspension, pare-brise, etc.) en les vidant de leurs technologies mécaniques antérieures. Cet exemple est pour eux emblématique de la propriété de l'informatique d'être une *technologie dissolvante* qui reconfigure en profondeur toutes les technologies et tous les métiers, voire les relations sociales et politiques[10]. Ce faisant, l'informatique se dissout elle-même comme sujet d'étude, rendant ainsi impossible à l'avenir une chronique unifiée de ses prochaines transformations :

> Au début des années 2020, l'histoire de l'informatique moderne a atteint le début de sa fin. La dernière grande histoire du progrès technologique moderniste se fracture finalement dans un chaos postmoderne. L'informatique est profondément imbriquée dans les évolutions structurelles essentielles des relations mondiales, de l'économie, de la société et de la culture. […] L'ordinateur étant devenu une machine véritablement universelle, l'histoire de l'informatique fait désormais partie de l'histoire de toute chose[11].

Protéiforme, obsolète, invisible, la programmation ne serait-elle pas tout simplement une *mauvaise idée* ? Ce chapitre va creuser davantage ce questionnement, en montrant la difficulté à cerner aussi bien ses pratiques que sa définition. Nous continuons cette section introductive en tentant de l'appréhender par ses aspects les plus positifs, ceux que mettent en avant les statistiques économiques, apparemment en mesure de dissiper aisément tous ces doutes, étant donné la place majeure que le développement logiciel occupe aussi bien dans la valeur ajoutée (§ 26) que dans l'emploi (§ 27) de nos économies.

§ 26. Une contribution économique majeure et diffuse

Lorsqu'on prend une vue d'ensemble de l'industrie informatique et des télécommunications, on est d'abord frappé par sa taille gigantesque. Avec une valeur ajoutée de l'ordre de 5 600 milliards de dollars américains en 2021 à l'échelle

10. (Haigh et Ceruzzi 2021, p. 423).
11. (ibid., p. 423).

mondiale[12], elle est plus importante que bien d'autres secteurs industriels, comme par exemple l'automobile (2 900 milliards de dollars) ou la pharmacie (1 400 milliards de dollars)[13].

Au sein de cette industrie, le développement logiciel ne représente néanmoins qu'un peu moins de 20% de la valeur ajoutée. La sous-traitance d'opérations* et de support de systèmes logiciels compte pour environ 14%, portant à un tiers la part totale du logiciel dans l'industrie. L'industrie des machines – qui va des composants semi-conducteurs jusqu'aux centres d'hébergement informatiques, privés ou en nuage*, en passant par les ordinateurs, tablettes et téléphones, compte pour un peu plus de 40%. Les télécommunications, qui sont devenues inséparables de l'informatique avec l'avènement d'Internet, comptent pour le quart restant de cette valeur ajoutée. En additionnant ces deux derniers ratios, on voit que deux tiers de l'industrie reste rivée aux machines et aux infrastructures physiques qui hébergent les programmes et les processus informatiques, ou permettent leur mise en relation.

Cependant, cette statistique doit être nuancée. D'abord, elle ne tient compte que des activités de développement faisant l'objet d'un échange marchand. Toutes celles qui sont réalisées pour les besoins propres des entreprises ne s'y trouvent pas. Or la part entre développement, opérations et infrastructures, parmi les équipes informatiques des entreprises, est respectivement de 30%, 50%, et 20% en moyenne[14], bien que ces ratios varient avec le taux de sous-traitance. L'infrastructure est en effet largement sous-traitée, ce qui est moins le cas des opérations et du développement logiciels.

Ensuite, cette statistique ne tient pas compte du fait que les opérations et le support logiciel sont souvent difficiles à distinguer de la programmation. Ces équipes peuvent parfois intervenir directement sur le code pour y apporter des correctifs mineurs. Surtout, dans leur rôle essentiel de configuration et de supervision des logiciels et des bases de données, qu'on appelle l'administration système*, elles ont le contrôle immédiat et tangible des machines, ce qui est bien, dans l'angle de vue large qui est le nôtre, de la programmation. Elles font d'ailleurs très souvent appel au scripting*, c'est-à-dire à la conception de séquences d'instructions

12. Statista, *Future-facing insights into technology markets*, consulté le 12 janvier 2023.
13. *idem*.
14. Il n'existe pas à notre connaissance de statistiques publiques à ce sujet. Ce ratio est dérivé de notre expérience professionnelle et a été confirmé par des spécialistes de l'industrie informatique.

automatisant ces tâches de gestion informatique. Cela vaut également pour la gestion de réseaux et de centres d'hébergement informatiques. Les opérateurs d'informatique en nuages* sont ainsi essentiellement tournés vers la programmation, en l'occurence de systèmes d'exploitation* d'infrastructures distribuées.

Certains fabricants de machines développent par ailleurs des logiciels qu'ils incluent dans leur produit final. Un exemple bien connu est Apple, qui vend ses ordinateurs et ses téléphones équipés d'une très riche suite d'applications. Dans tous ces cas de figure, la contribution de la programmation dans les statistiques industrielles est masquée, car elle apparaît sous les rubriques « machines », « infrastructures » ou « opérations » plutôt que sous celle de « développement logiciel ».

Plus fondamentalement encore, la conception de machines informatiques et de semi-conducteurs est aujourd'hui largement assimilée à un travail de programmation. Il n'est plus question en effet de dessiner des plans de circuits électroniques à la main, étant donné le nombre des composants et la complexité de leurs relations. Des langages de programmation spécifiques ont été conçus, comme VHDL ou Verilog, qui permettent de combiner logiquement des composants standard (nommés FPGA) ou des portes logiques élémentaires en un circuit intégré, et de déterminer jusqu'à l'organisation physique de ce dernier.

La statistique concernant la valeur ajoutée ne tient pas compte non plus des développements de logiciels réalisés sous une licence de logiciel libre. Or ceux-ci ont une part de marché dominante (de 50% à 90%) dans certaines catégories de logiciel, notamment les systèmes d'exploitation comme Linux ou Android, et les technologies web[15]. On associe souvent le logiciel libre à des motivations idéalistes, comme celles de Richard Stallman qui écrivit la première licence libre GNU, ou personnelles, comme celles de Linus Torvalds, le concepteur de Linux, qui déclarait :

> Ce monde serait bien meilleur si les gens avaient moins d'idéologie, et beaucoup plus de *« je fais ça parce que c'est AMUSANT et parce que d'autres pourraient le trouver utile, pas parce que j'ai une religion »*[16].

Cependant, de manière plus prosaïque, la participation non rémunérée à un projet open-source* est aussi une manière pour un développeur de se former (au début de sa carrière et de manière continue), de construire un réseau professionnel et d'augmenter sa valeur sur le marché par des contributions réussies[17]. Surtout, comme

15. (Schrape 2019, p. 7).
16. (Torvalds 2002).
17. Voir par exemple les remarques à ce sujet de ce blog, et de (Larribeau 2019, p. 66-67).

le montre Schrape (2019) les projets open-source* sont de plus en plus financés et dirigés par des entreprises, qui trouvent dans ce modèle de nombreux avantages en termes de collaboration inter-entreprise, d'accès à l'innovation, et de réduction des coûts de développement.

Ce désintérêt de certaines entreprises pour la propriété intellectuelle du code logiciel développé s'explique également par leur modèles d'affaires. L'entreprise RedHat est célèbre pour avoir réussi la première, au début des années 2000, à devenir rentable grâce au système d'exploitation Linux. Tout en distribuant gratuitement le logiciel aux entreprises, elle leur facturait des services de maintenance, de support, de formation et de conseil. Cette époque voit aussi le début des modèles ASP*, aujourd'hui connus sous l'acronyme SaaS*, qui permettent à des clients particuliers ou à des organisations d'utiliser les fonctionnalités d'un logiciel hébergé sur des serveurs distants accessibles par Internet.

Ces modèles sont la raison de plus en plus importante de la sous-représentation du développement logiciel dans les statistiques économiques. Par exemple, une entreprise comme PayPal est considérée comme un intermédiaire financier, cependant son actif cœur est le logiciel et sa capacité d'innovation informatique. Il en est de même de toutes les entreprises qui proposent des services financiers en ligne et qu'on range sous l'acronyme de la *FinTech* : le développement informatique est une composante critique de leur activité, cependant elles ne le valorisent qu'indirectement.

Ce modèle d'affaires n'est bien sûr pas spécifique à l'industrie financière. Les services numériques se développent également dans la santé – par exemple l'hébergement de dossiers médicaux personnalisés, ou le suivi de données vitales comme le rythme cardiaque. La société Amazon a construit son hégémonie dans la grande distribution en se concevant dès le départ, selon les mots de son fondateur Jeff Bezos, non comme un distributeur, mais « comme une entreprise de technologie visant à simplifier les transactions en ligne des consommateurs[18]. ». Plus récemment, reprenant ce leitmotiv, le célèbre multi-entrepreneur Elon Musk a décrit sa société Tesla qui fabrique des voitures électriques, comme « une entreprise de logiciel aussi bien que de matériel[19] ». On a là des exemples frappants d'entreprises dont le développement logiciel est une compétence cœur, mais qui ne la valorisent qu'indirectement, échappant ainsi aux statistiques concernant la valeur ajoutée

18. (M. Hall [2008] 2017).
19. (Haigh et Ceruzzi 2021, p. 411).

citées plus haut.

Tous ces arguments convergent vers le constat suivant : il est difficile de cerner la contribution de la programmation dans les statistiques économiques car elle est prégnante non seulement dans les opérations et les infrastructures informatiques, mais elle infiltre plus généralement de plus en plus d'activités économiques. Par ailleurs, grâce à l'open-source*, il existe un continuum entre développement professionnel et amateur, médiatisé notamment par les étudiants en formation ou aspirant à débuter leur carrière.

§ 27. Une profession aux caractères uniques

Ces constats sont confirmés par l'étude des statistiques de l'emploi. Comme celles concernant la valeur ajoutée, elles montrent d'abord que le développement n'est pas l'activité principale des informaticiens. Aux États-Unis, où elles sont disponibles avec un détail fin, parmi les 4,6 millions d'informaticiens, seuls 40% (soit 1,8 millions) sont des développeurs. 30% travaillent dans les opérations* et le support informatique aux utilisateurs, 15% dans les réseaux informatiques et la sécurité, et les 15% restants dans des activités diverses comme l'administration de bases de données, la conception de sites Web ou le numérique en général. Ces populations sont en forte croissance, de l'ordre de 15% par an en moyenne, et 25% pour les développeurs, contre une croissance totale de l'emploi de 5 à 7% par an[20].

Il existe cependant plusieurs manières d'interpréter ces ratios. Pour l'ACM, la principale association d'informaticiens. ils prouvent la fausseté de ce que Peter Denning, son président pendant plusieurs années, appelle « le mythe Informatique = Programmation ». Casser ce « mythe » est important pour l'ACM depuis le début des années 2000 afin d'encourager les jeunes étudiants à se diriger vers une profession en manque chronique de main d'œuvre.

> La plupart des gens ne voient pas les histoires des architectes informatiques, des ingénieurs réseau, des ingénieurs en systèmes d'exploitation, des ingénieurs en bases de données, des graphistes, des architectes logiciels, des concepteurs de systèmes logiciels, des experts en sécurité, en simulation, en réalité virtuelle, en supercalculateurs, les roboticiens, et bien d'autres encore. Toutes les histoires sont racontées comme si elles avaient été faites par des programmeurs[21].

20. Source : site du *U.S. Bureau of Labor Statistics*.
21. (Denning 2004, p. 16).

Ces constats sont bien sûr corrects. Cependant, si on entend *programmation* au sens large que nous avons suggéré plus haut, celui de « diriger des machines à faire ce que l'on veut », tous ces métiers peuvent être dits avoir affaire essentiellement à de la programmation de machines. Comme nous l'avons déjà dit au développement précédent et également en introduction[22], même si l'interaction de ces informaticiens avec les machines ne prend pas la forme du codage associée traditionnellement à la programmation, elle a essentiellement affaire au contrôle de celles-ci, et par là est souvent amenée à prendre des formes similaires, comme le scripting*.

Dans cette seconde interprétation, de telles statistiques montrent plutôt la fragmentation des modalités par lesquelles le contrôle des machines peut être effectué.

À cette impression de fragmentation s'ajoute celle d'une ouverture du métier d'informaticien, lorsqu'on tient compte du fait qu'il y aurait autant de personnes ayant des activités informatiques complexes dans le cadre de leur occupation professionnelle principale, que d'informaticiens au sens strict. Ce rapport de 1 à 2 est confirmé par une étude récente qui estime à 4,3 millions le nombre de développeurs aux États-Unis, contre les 1,8 millions de développeurs professionnels recensés par le *Bureau of Labor Statistics* américain[23]. Le rapport de développeurs informaticiens aux développeurs non-informaticiens serait donc ici de 1 à 2,4. L'écart avec la première étude concluant au rapport de 1 à 2 peut s'expliquer par le fait que cette dernière est fondée sur des chiffres de 2012, et il est logique que le rapport ait augmenté depuis, avec la diffusion croissante des techniques informatiques dans les pratiques professionnelles, comme l'affirment Muro, Liu et S. Kulkarni (2017). Il peut aussi s'expliquer par le fait que parmi les professionnels non informaticiens, les activités de développement doivent être sur-représentées par rapport aux autres activités informatiques, avec sans doute également les activités numériques et de gestion des données. D'après cette étude, il y a 27 millions de développeurs dans le monde en 2022, en incluant ceux dont l'occupation professionnelle principale n'est pas le logiciel, comme par exemple :

— Des ingénieurs électriciens, qui conçoivent, développent, testent et fabriquent des équipements électriques.
— Des ingénieurs en matériel informatique, qui conçoivent, développent, testent

22. Voir plus haut, § 21.
23. Source : Étude de Evans Data Corporation, citée par (Warren 2021).

et fabriquent des composants informatiques.
— Des concepteurs en CAO*, qui […] conçoivent des composants industriels. […]
— Des économistes, qui étudient le développement, la distribution et l'utilisation des biens et des services.
— Des mathématiciens, qui analysent les données et appliquent des modèles mathématiques pour résoudre des problèmes de gestion (*business problems*).
— Des rédacteurs techniques, qui créent des instructions et des diagrammes pour les guides d'utilisation et d'instruction[24].

On retrouve dans les deux premières catégories mentionnées les ingénieurs de matériel informatique dont nous avons vu plus haut que le travail s'apparentait effectivement à de la programmation. Mais plus généralement la programmation déborde les frontières de l'informatique : les ingénieurs programment de plus en plus pour concevoir des pièces mécaniques, les chimistes pour composer les matériaux ayant les propriétés souhaitées, les banquiers d'affaires pour optimiser leurs modèles d'investissement, etc.

Une autre étude[25] estime plutôt à un tiers la part de développeurs non professionnels, mais avec une méthodologie différente, basée sur l'intensité de l'activité de développement, telle qu'on peut notamment l'observer sur les plateformes comme Github. Les professionnels ne programmant qu'occasionnellement ou par à-coups ne sont donc pas comptabilisés. Par contre, cette étude met en lumière l'importance du développement réalisé pour des motifs *non professionnels*, en particulier celui des amateurs intensifs et des étudiants que l'open-source* permet d'intégrer à des projets significatifs.

Au-delà de ces programmeurs intensifs – qui comptent donc, aux États-Unis, pour 3 à 5% de la population active, et pour 1 à 3% de la population totale, gravite un cercle croissant de personnes qui programment occasionnellement ou qui savent réaliser des tâches complexes sur ordinateur. Selon Eurostat, l'organisme statistique de l'Union Européenne, environ 6% des résidents de l'Union avaient écrit un programme au cours des trois derniers mois précédent l'enquête[26], et un peu plus d'un quart des résidents ont des compétences numériques « plus qu'élémentaires » (*above basic digital skills*), ce qui ne veut pas dire qu'ils savent pro-

24. (ibid.).
25. (Shuermans et Voskoglou 2019, p. 6).
26. Eurostat, base de données *Compétences numériques des particuliers*, fichier isoc_sk_dskl_i21, consulté le 25 janvier 2023.

grammer[27]. Une étude internationale ciblant des jeunes de 15 ans montre qu'environ la moitié ont ce type de compétences[28].

Il est donc difficile de cerner l'activité de la programmation car, en partant du développeur professionnel, en allant vers l'étudiant pré-professionnel, l'amateur acharné, le programmeur occasionnel, l'utilisateur qui en sait assez pour « lire » un programme, et enfin celui qui, tout en ne sachant pas programmer au sens strict, peut monter un site web, gérer une base de données, ou réaliser des tâches complexes similaires, il existe qu'un continuum que les statistiques ont de la peine à profiler, d'autant plus que la série de ces cercles concentriques croît rapidement.

Au terme de ce rapide survol statistique, la programmation apparaît doublement insaisissable, fragmentaire et ouverte à la fois. D'une part, si elle est la pratique professionnelle centrale dans l'industrie de l'informatique et des télécommunications, elle s'y fragmente en une multitude de métiers et s'infiltre dans les pratiques professionnelles d'autres industries qui utilisent l'informatique de manière de plus en plus intensive. D'autre part, entre pratique professionnelle et pratique occasionnelle et amateur, il y a un continuum dont l'open-source* constitue le liant.

Il est important de bien saisir le caractère singulier de cette situation, qui nous semble unique dans le monde du travail, comme cela se voit si on compare l'informatique à d'autres professions intellectuelles, la médecine, le droit, l'ingénierie ou les mathématiques.

La profession de médecin est strictement encadrée par la législation, qui requiert la sanction de certains diplômes pour permettre à une personne d'exercer les pouvoirs associés à cette profession (établir la prescription de certains médicaments, par exemple). Il y a certes des professions para-médicales, mais celles-ci sont clairement distinguées dans leurs droits d'exercice. S'il y a des médecins du travail dans des entreprises diverses, notamment industrielles, ceux-ci sont aussi clairement identifiés comme membres du corps médical.

Les professions du droit sont plus ouvertes. Aux côtés des juges et des avocats, on trouve dans toutes les industries des juristes qui s'occupent des affaires légales de leurs entreprises ou administrations. Par ailleurs, des professionnels de métiers divers sont parfois formés aux rudiments du droit. Ils savent ainsi établir ou vérifier des contrats – on pourrait avancer ici l'idée qu'un contrat est l'équivalent d'un

27. *idem*, fichier isoc_sk_cskl_i21.
28. (Fraillon et al. 2020).

programme en informatique[29]. Cependant, on ne trouve pas une communauté active d'amateurs contribuant à des projets collectifs de droit.

Le cas des différentes disciplines d'ingénierie est proche de celui du droit. Aux côtés de l'industrie de la construction, de nombreux autres métiers emploient également des ingénieurs. Cependant, contrairement au droit, il est plus difficile de former des professionnels experts d'autres disciplines à des rudiments d'ingénierie. S'il peut être aisé, après une courte formation, d'évaluer la qualité d'un contrat, il est beaucoup plus difficile de savoir si un bâtiment ou une pièce mécanique sont bien conçus. Dans le cas de la construction civile, on peut mettre en avant les « ingénieurs amateurs » que sont les bricoleurs capables de réaliser des travaux chez eux, voire de monter des petites structures de bâtiments, et cela parfois de manière collaborative. Mais aucune collaboration de grande ampleur ne peut être observée entre ingénieurs professionnels et bricoleurs, fussent-ils étudiants-ingénieurs. Tout bâtiment d'une certaine taille et complexité est soumis à des régulations et des certifications qui, comme en médecine, écartent la possibilité de l'amateurisme.

La programmation pourrait être enfin rapprochée des mathématiques, en ce que celles-ci s'infiltrent dans de très nombreux métiers et pratiques professionnelles. On pourrait aussi arguer que les mathématiques sont une discipline par essence ouverte à l'amateurisme, puisqu'elles ne sont contraintes ni matériellement ni juridiquement, même si la population de mathématiciens amateurs est très restreinte. Cependant, les mathématiques n'ont pas donné naissance à une industrie marchande, et sont restées une discipline de recherche. On pourrait mettre en avant les bureaux d'études spécialisés en mathématiques appliquées, qui résolvent des problèmes pour les ingénieurs, les logisticiens, les financiers, etc. Or justement, ces services prennent aujourd'hui la forme informatique de modèles de simulation, de calcul, ou de traitement de données.

Ce dernier rapprochement pointe cependant vers une similarité qui a été souvent remarquée entre l'informatique et les mathématiques, si bien qu'on a pu qualifier la première d'*ingénierie des mathématiques*[30]. Cette expression est selon nous révélatrice du caractère mystérieusement hybride de l'informatique, qui peut emprunter aux disciplines d'ingénierie leurs pratiques professionnelles et l'aspect tangible de leurs productions mais également aux mathématiques la liberté de la recherche et la fluidité des formes d'expression.

29. Le lien étroit entre informatique et institutions humaines sera évoqué au chapitre 12.
30. (Hartmanis 1995, p. 14).

Nous allons, dans les sections suivantes, étudier successivement ces deux aspects de la programmation afin d'essayer de cerner sa nature. D'une part on peut tenter de la caractériser *axiologiquement* en observant ses pratiques professionnelles, puisque celles-ci devraient représenter la programmation « à son meilleur », les pratiques des amateurs et des non-spécialistes devant être évaluées relativement aux normes d'excellence qui sont activement définies et sans cesse améliorées par les praticiens professionnels. D'autre part, on peut tenter de la définir *analytiquement* en prenant acte de cette diversité et en tentant d'en trouver le schéma minimal commun qui nous permet de distinguer la programmation de toute autre pratique.

2.2 L'exigence de systématicité

L'objectif de cette section et de la suivante est d'étudier la programmation de manière axiologique, c'est-à-dire à partir des « bonnes pratiques » que promeut la programmation professionnelle. Dans cette section, il s'agit de rendre compte, de manière préliminaire, des raisons internes de sa professionnalisation. Nous commençons par décrire une expérience élémentaire de programmation (§ 28) qui met en évidence une *exigence de systématicité* difficile à satisfaire sans la discipline d'une méthode (§ 29). Nous justifions d'abord cette exigence d'un point de vue pragmatique, en évoquant la notion d'*obsolescence informatique* (§ 30) puis tentons de l'illustrer par les motivations de la *revue de code*, une technique courante d'assurance-qualité dans le développement logiciel (§ 31).

§ 28. Exemple d'une programmation d'amateur

L'objectif de ce développement est de donner à la lectrice qui n'est pas versée dans la programmation un aperçu de cette activité à son niveau le plus élémentaire – qui se trouve heureusement être aussi celui de l'auteur. Pour ce faire, nous allons ici exposer notre propre cheminement lors d'un petit exercice de programmation, choisi à dessein pour illustrer notre réflexion. Bien entendu, une telle expérience n'est pas scientifiquement significative – il ne s'agit pas de faire une psychologie naïve de la programmation – mais simplement de décrire la manière dont nous-même l'expérimentons, afin que la lectrice puisse apprécier notre propre point de départ. Il ne s'agit pas non plus de présenter une *bonne* manière de programmer, avec une visée normative. Au contraire, nous tentons de montrer les hésitations

et défaillances de notre réflexion, afin de les contraster plus loin avec les bonnes pratiques de la profession.

Nous avons choisi pour illustration un problème que *nous n'avions pas déjà résolu* par le passé, afin de pouvoir observer nos propres réflexions et difficultés au fur et à mesure qu'elles survenaient, et non par reconstruction *a posteriori*. Nous nous sommes limités à un problème simple à expliquer et à résoudre, afin que l'exposé ne soit pas trop long. Pour la même raison, nous avons programmé en langage Python, car sa syntaxe est parmi les plus simples à expliquer.

Le problème en question est un de ces exercices classiques par lesquels on apprend à programmer, un problème bien posé et donc simple à modéliser[31]. Il y a bien sûr là un autre biais important, sur lequel nous reviendrons souvent dans le cours de la réflexion. D'une certaine manière cependant, ce biais fait également partie de notre point de départ, puisqu'il faut partir de ce qu'on entend *généralement* par la programmation, et les gens se rappellent de ce qu'ils en ont appris au lycée ou à l'université.

Le problème est le suivant : étant donné un système de pièces de monnaies (par exemple les pièces jaunes et roses actuelles de 1, 2, 5, 10, 20, 50 centimes), de combien de manières est-il possible de faire l'appoint d'une somme donnée ? Par exemple, quelques appoints de 1 euro avec les pièces ci-dessus, parmi les 4562 possibles, sont :

1. 2 pièces de 50 centimes.
2. 1 pièce de 50 centimes, 2 pièces de 20 centimes, 1 pièce de 10 centimes.
3. 1 pièce de 50 centimes, 2 pièces de 20 centimes, 1 pièce de 5 centimes, 5 pièces de 1 centime.
4. 5 pièces de 20 centimes.
5. etc.

Cette question fait partie du champ des problèmes du *rendu de monnaie* (*change-making problems*), dont certains sont encore non résolus.

Le problème qui nous occupe est par contre très simple. Nous allons décrire notre raisonnement personnel puis donner les versions successives du programme tel que nous l'avons écrit, en passant donc à la première personne du singulier pour cette partie de la réflexion. La lectrice impatiente peut aller directement aux

31. Nous l'avons d'ailleurs trouvé dans l'ouvrage classique de (Abelson, G. J. Sussman et J. Sussman 1985).

conclusions de cet exposé quelque peu fastidieux et laborieux d'une *expérience de pensée*, au sens propre du terme.

Mon intuition élémentaire, à l'exposé du problème, fut que celui-ci avait une structure récursive, c'est-à-dire que la liste des solutions pour un système de pièces donné allait être exprimé en fonction de ses solutions pour des restrictions de ce même système. Par exemple, les manières de faire l'appoint avec 1, 2 et 5 centimes allait être une expansion des manières de le faire avec 1 et 2 centimes.

Je commençai donc à explorer cet exemple avec un papier et un crayon. L'objectif était de trouver la formule centrale de la récursivité qui formerait le cœur du programme. Je regardai les manières de faire l'appoint avec 1, 2, et 5 centimes sur 5, 10 et 15 centimes. Cette réflexion dura une bonne dizaine de minutes. Mon blocage était le suivant : si on part de la décomposition de 10 en 5 + 5, étant donné que 5 peut être sommé de trois manières avec des 2 et des 1 (2x2+1, 2+3x1, 5x1), comment calculer les sommes possibles de 10 à partir de 2 et de 1 ? Je voyais bien que je ne pouvais multiplier les deux combinaisons, car j'aurais eu des doublons. N'aimant pas le calcul combinatoire, je me trouvais un peu bloqué.

L'intuition de la solution me vint en considérant la notion de *reste*. Pour faire l'appoint avec un système de pièces, je dois choisir une pièce, puis voir les manières de faire l'appoint sur le reste de la somme. Je peux alors choisir la même pièce à nouveau (si la somme reste plus grande que sa valeur) ou me débrouiller avec les pièces restantes. Par exemple, avec 15 centimes, je peux choisir de commencer l'appoint avec 5 centimes. Le problème est alors de faire l'appoint sur 10 centimes. Je peux à nouveau choisir 5 centimes (et regarder les manières de faire l'appoint sur les 5 centimes restants) ou chercher les manières de faire l'appoint de 10 centimes uniquement avec des pièces de 1 et 2 centimes.

Ainsi, la récurrence porte sur des problèmes P fonction d'une somme S sur laquelle faire l'appoint et d'un système de pièces $[c_1, c_2, ..., c_n]$. La formule importante était la suivante :

$$P(S, [c_1, c_2, ..., c_n]) = P(S - c_1, [c_1, c_2, ..., c_n]) + P(S, [c_2, ..., c_n])$$

Cette récurrence, en d'autres termes, a une structure arborescente, et la formule ci-dessus exprime l'embranchement possible à chaque nœud : soit choisir la première pièce qui se présente dans la liste et explorer le problème de l'appoint sur la différence (tout en gardant la pièce dans le jeu), soit mettre de côté cette pièce et explorer le problème sur la même somme, mais avec le système de pièces restreint. Un petit raisonnement m'assura qu'il n'y aurait pas de doublon dans une

telle démarche. Par contre, je n'étais pas sûr que *l'ordre* de présentation des pièces ne comptât pas, et je choisis de commencer à programmer avec une liste de pièces ordonnées de manière décroissante (50, 20, 10, 5, 2, 1), car c'est ainsi qu'il me serait le plus naturel de vérifier le bon fonctionnement de mon programme.

Il faut remarquer ici que je n'écrivis pas la formule telle que je la présente ci-dessus : *je la codai directement*. C'est à ce stade en effet que je lâchai papier et crayon pour me mettre face à l'ordinateur. J'écrivis assez vite le code suivant :

CODE 2.1 – Programme Python sur le rendu de monnaie, V1

```
1  def find_Change (money,coins):
2      # le résultat de cette fonction est une liste des solutions,
3      # qui sont elles-mêmes des listes de pièces (liste de liste)
4      if money == 0: return [[]]
5      if coins == []: return []
6      x = coins[0]
7      if (money >= x):
8          return (append (x,find_Change(money-x,coins)) \
9             + find_Change (money,coins[1:]))
10     else: return find_Change (money,coins[1:])
11
12 def append (elem,lstlst):
13     #fonction qui ajoute un élément (ici une pièce)
14     #à toutes les listes d'une liste (ici la liste de solutions)
15     def f(x): return [elem]+x
16     return list (map(f,lstlst))
```

Il y eut bien sûr plusieurs allées et retours, surtout du fait d'erreurs de syntaxe. L'instruction plus difficile à écrire fut celle des lignes 7 à 10 qui capture la formule ci-dessus. Mais avant cela, il fallait déterminer *le type* de ma fonction de récurrence (ligne 1). Son argument n'était rien d'autre que le problème à résoudre, une paire constituée d'une somme sur laquelle faire l'appoint (`money`, un entier naturel) et d'un système de pièces disponibles (`coins`, une liste d'entiers naturels). Son résultat devait être la solution du problème, c'est-à-dire une liste de combinaisons possibles de pièces énumérées dans la liste initiale, et dont la somme était l'appoint demandé. Chaque combinaison se représentant naturellement comme une liste de pièces, le résultat devait donc être une liste de listes d'entiers. Comme Python, contrairement à d'autres langages, n'oblige pas à prédéfinir sur quels types de données porte une fonction, j'explicitai naturellement ce choix non évident dans les commentaires lignes 2-3. On voit par exemple en 2.2 ce que donne l'exécution

du programme sur un problème simple, où la première ligne correspond à l'appel du programme par l'utilisateur (après l'invite de commande* que nous représentons par >>>), et la seconde à son résultat.

CODE 2.2 – Exécution du programme :
manières de rendre la monnaie sur 5€ avec des pièces de 1 et 2€.

```
1 >>> find_Change(5, [2,1])
2 [[1, 1, 1, 1, 1], [1, 1, 1, 2], [1, 2, 2]]
```

Les lignes 4 et 5 du programme fixent les points de terminaison de la récurrence. Je les écrivis immédiatement après la réflexion ci-dessus, car ils m'aidèrent à m'assurer que j'avais fait les bons choix. Si l'appoint restant à faire est 0, c'est que les opérations antérieures ont réussi, et il faut donc renvoyer le singleton d'une liste vide. Dans le cas contraire, s'il n'y a plus de pièces disponibles, c'est que le problème n'a pas de solution, il faut donc renvoyer la liste vide. Ces deux terminaisons sont donc diamétralement opposées, l'une signifiant le succès et l'autre l'échec de l'exploration d'une combinaison, quoiqu'elles s'écrivent de manière presque similaire.

Enfin arrive la formule centrale. Elle dépend de savoir si je peux utiliser la première pièce pour faire l'appoint c'est-à-dire si l'appoint à faire est supérieur à la valeur de la pièce. Si ce n'est pas le cas, (ligne 10, `else`), je la soustrais de la liste de pièces disponibles (ce qui se fait en Python grâce à l'expression `coins[1:]`) et je lance la résolution de ce nouveau problème. Si c'est le cas (lignes 8-9) deux possibilités s'ouvrent à moi, comme indiqué sur la formule ci-dessus, qui tient ici sur deux lignes, utiliser la pièce (ligne 8) ou non (ligne 9). Cette deuxième possibilité revient au problème précédent : les lignes 9 et 10 sont presque identiques, et il doit donc être possible de simplifier cette expression. La ligne 8 présente une apparence compliquée, car je dois ajouter la pièce choisie aux solutions donnant l'appoint du reste. Par exemple, si j'utilise une pièce de 5 centimes pour faire l'appoint de 8 centimes, et qu'une solution pour faire l'appoint sur le reste de 3 centimes est [2, 1], alors la solution correspondante pour 8 centimes est [5, 2, 1]. Cette apposition technique de la pièce ajoutée aux solutions d'appoint renvoyées par le problème résiduel est opéré par la fonction `append`, qui est définie lignes 13 à 18, et qui n'a pas d'intérêt pour notre propos.

Il est à noter qu'une erreur importante que je fis dans le codage de ces formules fut – outre les erreurs de syntaxe – d'appliquer cet ajout de la pièce considérée aux deux branches des solutions possibles, ce dont je ne m'aperçus qu'à l'exécution.

Il s'agissait là d'une erreur *conceptuelle*, liée au fait que je me mis à coder assez rapidement, en n'ayant qu'une idée intuitive de la solution.

Une fois la fonction réalisée, je testai qu'elle fonctionnait bien sur quelques exemples. Par exemple, il y a 4562 manières de faire l'appoint d'1 euro avec les 6 pièces jaunes et roses du système européen. Ensuite je commençai à optimiser le code. Je le récrivis notamment pour me débarrasser de la redondance observée entre les lignes 8 et 10. Cela se fit en remarquant que je pouvais tout à fait laisser la fonction explorer les cas où l'appoint à réaliser était inférieur à la pièce choisie, en en faisant tout simplement une seconde condition de terminaison négative. Cela permit de rendre la fonction plus lisible en ôtant 4 lignes de code :

CODE 2.3 – Programme Python sur le rendu de monnaie, V2

```
1  def find_Change (money,coins):
2      # le résultat de cette fonction est une liste des solutions,
3      # qui sont elles-mêmes des listes de pièces (liste de liste)
4      if money == 0: return [[]]
5      if coins == [] or money <0: return []
6      return (append (coins[0],find_Change(money-coins[0],coins)))\
7             + find_Change (money,coins[1:])
8
9  def append (elem,lstlst):
10     #fonction qui ajoute un élément (ici une pièce)
11     #à toutes les listes d'une liste (ici la liste de solutions)
12     def f(x): return [elem]+x
13     return list (map(f,lstlst))
```

Mon second travail fut d'optimiser la récurrence. Je savais qu'elle était sans doute très inefficace dans l'exploration de l'arbre des combinaisons possibles, puisqu'elle recalculait systématiquement tous les problèmes, y compris les plus simples, qui devaient être aussi les plus fréquents, comme trouver l'appoint de 5 centimes avec des pièces de 2 et 1 centimes.

Ici j'utilisai une technique d'optimisation que je connaissais déjà : en mémorisant progressivement dans une table les solutions des problèmes déjà résolus, il serait possible d'éviter de lancer leur calcul à nouveau, en consultant d'abord cette table. Cela passait par la distinction d'une fonction-maître et d'une fonction-esclave s'appelant l'une l'autre pour former la récurrence. Avant cela, il fallait définir la base de solutions. La structure naturelle pour ce faire en Python est le dictionnaire `dict()`, qui permet d'associer des clés à des valeurs. Ici mes clés

étaient les problèmes, dont le type était la paire d'un entier et d'une liste d'entiers, et les valeurs les listes de combinaisons possibles, dont le type était une liste de listes d'entiers. Voici le code auquel j'aboutis :

CODE 2.4 – Programme Python sur le rendu de monnaie, V3

```
1  k_base = dict() # base de solutions
2
3  def find_Change (money,coins):
4      #fonction principale à appeler
5      #si le pb n'est pas dans la base,
6      #calculer sa solution:
7      idc=id_c(coins)
8      if (money,idc) not in k_base :
9          k_base[(money,idc)] = calc_Change(money,coins)
10     #en tous cas, retourner la solution
11     #qui se trouve dans la base:
12     return k_base[(money,idc)]
13
14 def calc_Change (money,coins):
15     #fonction inchangée, sauf qu'elle
16     #ne s'appelle plus elle-même:
17     if money == 0: return [[]]
18     if coins == [] or money <0: return []
19     return (append (coins[0],find_Change(money-coins[0],coins)))\
20            + find_Change (money,coins[1:])
21
22 def append (elem,lstlst):
23     def f(x): return [elem]+x
24     return list (map(f,lstlst))
25
26 def id_c (coins):
27     #fonction à réécrire
28     return sum(coins)
29
30 a = find_Change (100,[50,20,10,5,2,1])
31 b = a[0:15]
32 c = ' '.join(str(e) for e in a[4314])
```

La ligne 1 définit le dictionnaire `k_base`, qui est une variable globale : elle persiste à travers l'appel successif des fonctions, qui l'enrichissent successivement. La fonction principale, à présent, (lignes 3-11) se contente de rendre le résultat qu'elle trouve dans `k_base`, et si elle ne l'y trouve pas, de lancer son calcul par l'appel à

la fonction définie auparavant, qui s'appelle à présent `calc_Change()`. Celle-ci est inchangée, sauf qu'au moment de la récurrence (lignes 18-19) elle ne s'appelle plus elle-même, mais bien la fonction principale. C'est ce mécanisme croisé qui permet d'optimiser le calcul.

Il convient ici de mentionner un problème technique dont je m'aperçus en faisant des essais sur le dictionnaire. Python n'accepte pas des clés composées de listes, sans doute pour de bonnes raisons. Je dus donc trouver un moyen de représenter mes listes de pièces autrement, ce que je fis en prenant tout simplement leur somme, une solution très mauvaise car elle n'évite pas les doublons éventuels, qui seraient catastrophiques pour l'exactitude des résultats. Il s'agissait là d'un pis-aller fait par impatience, dont je savais qu'il fonctionnerait *sur les exemples* que je voulais tester.

Pour étudier les résultats et comparer les deux versions de mes fonctions, je n'utilisai pas d'outil d'analyse (ce qui est la bonne pratique), mais j'introduisis des compteurs dans mes fonctions, et mis en place un chronomètre. Je ne reproduis pas ce code qui n'a pas grand intérêt, mais les résultats étaient significatifs : la fonction optimisée était en moyenne 15 fois plus rapide que la fonction récursive simple. Le dictionnaire mis en place avait en effet seulement 344 entrées pour l'appoint d'1 euro avec les 6 pièces jaunes et roses du système européen, tandis que la fonction simplement récursive était appelée environ 209 000 fois. Je vérifiai bien entendu que les deux fonctions donnaient toujours le même résultat sur des exemples différents (ce qui me tint lieu de preuve de correction) et je vérifiai également que l'ordre des pièces n'influait pas sur le résultat – par contre il influait sur la rapidité.

Au terme de cet exercice qui dura quelques heures, de nombreuses choses restaient à faire pour finaliser ce travail, notamment les suivantes :

1. Réécrire la fonction signature `id_c()` d'un système de pièces pour éviter tout doublon, tout en la gardant efficace.
2. Débarrasser le code de la variable globale `k`, qui empêcherait ma fonction d'être intégrée à un code plus large.
3. Améliorer l'interface utilisateur.
4. Vérifier formellement la correction de l'agorithme.
5. Calculer théoriquement et mesurer empiriquement la performance de l'algorithme sur différents jeux de valeur pour comprendre son comportement, dans ses deux versions.

§ 29. L'exigence de systématicité

L'exposé laborieux de cette expérience très simple a pour mérite de mettre en évidence plusieurs points utiles pour notre réflexion :

1) D'abord le problème lui-même a une forme typique : trouver *toutes* les solutions possibles d'un problème, ou *toutes* les situations possibles. La visée de la programmation est naturellement systématique. Pour ce faire, la récursivité (ou sa version simple, l'itération) semble être un outil également naturel, car elle tente de trouver une formule répétitive d'exploration systématique des possibles.
2) Le problème n'était jamais complètement *abstrait* dans notre esprit. Même si nous ne faisions que manipuler des entiers et des listes d'entiers, nous avions à l'esprit des pièces de monnaie, et cela aidait notre réflexion. Il est reconnu par les psychologues que les personnes résolvent mieux les problèmes lorsque ceux-ci sont posés sous la forme de représentations familières[32].
3) Une réflexion centrale est nécessaire pour trouver un fil directeur, ici la formule de récurrence. Cette réflexion a été ici inductive. Elle s'est déroulée avec papier et crayon. Découlant immédiatement de cette réflexion, vint également un travail de représentation, qui nous conduisit à définir comme objet fondamental le problème de l'appoint, et à le représenter par la paire d'un entier et d'une liste d'entiers. Ce choix fut mis à l'épreuve par des impossibilités techniques et dut être adapté.
4) Les deux idées principales qui nous sont venues à l'esprit pour résoudre le problème (la récursivité, la mise en table des solutions) avaient été apprises. Il s'agit de schémas couramment présentés dans les ouvrages didactiques. Nous ne savons pas si nous aurions trouvé ces « idées » tout seul. De la même manière, nous connaissions suffisamment Python, et les langages de programmation en général, pour savoir quelles possibilités de représentation des objets ils offraient, par exemple les structures de paire, de liste et de dictionnaire.
5) Le processus lui-même tout entier fut très hésitant et itératif, constitué de tentatives et d'erreurs, de tests et de corrections. Le programme fut réécrit au moins deux fois (en fait plus) à la recherche d'optimisation et de simplification.
6) Dans ce processus, il y eut des raccourcis peu recommandables (comme la fonction `id_c`) et l'usage de mauvaises pratiques (comme l'utilisation de compteurs pour évaluer la performance des fonctions). Tout en étant conscient de cela, on peut passer outre, car on cherche à aller à l'essentiel et on remet à plus tard la

32. (Hoffrage et al. 2002).

correction de ces défauts. On peut ainsi laisser des choses inachevées (comme ici une petite fonction à réécrire). Il est significatif que ces défauts apparaissent naturellement même dans un problème aussi simple que celui considéré.
7) De la même manière, on réalisa des vérifications extrêmement limitées de la correction et de la performance de son programme, sur la base de quelques exemples, à partir desquels on avait d'ailleurs raisonné.

Ces constats peuvent être rapprochés des conclusions des études de psychologie de la programmation. Ainsi Green (1990b, p. 131) affirme qu'elle se présente comme une activité de conception exploratoire, qui implique de nombreux allers et retours ; des observations détaillées montrent que les zones du programme les plus retravaillées (qu'on mesure par la fréquence d'usage de la touche *retour arrière*) sont celles requérant le plus de planification – ainsi par exemple la structure d'une liste de conditions. Pennington et Grabowski (1990, p. 46) décrivent cette exploration comme un va-et-vient entre composition du programme et compréhension du résultat obtenu, entre le *quoi* et le *comment* – les étapes de position du problème, de conception de la solution, de codage et de test étant par là étroitement solidaires. En particulier, contrairement aux recommandations de la programmation structurée de progresser par décompositions modulaires successives du problème à résoudre, les psychologues observent que les programmeurs commencent souvent par implémenter entièrement une solution cœur minimale avant de la compléter par incréments successifs[33] – comme nous l'avons fait nous-même.

Il y a donc une tension interne à la programmation entre d'une part, une visée systématique qui requiert l'intuition centrale d'une représentation et d'un « mécanisme » globaux, et d'autre part, tout le processus lui-même de découverte, de mise en place et de vérification de ces intuitions, qui n'est nullement systématique, mais intuitif, hésitant, approximatif, et qu'on constate dépendre à la fois de bons et de mauvais apprentissages.

On pourrait objecter que les déficiences du processus de pensée décrit ici ne tiennent qu'à la médiocrité du programmeur lui-même. Un programmeur professionnel jugerait certainement de manière très sévère les pratiques dilettantes que nous avons ici avouées. Il y a très certainement une *discipline de la programmation*, pour reprendre le titre d'un ouvrage célèbre de Dijkstra (1976), dont l'enjeu pourrait être formulé de manière suivante : *comment est-il possible de parvenir à la systématicité par une réflexion qui, laissée à sa pente naturelle, est tout sauf*

33. (Pennington et Grabowski 1990, p. 51).

systématique ?

§ 30. L'obsolescence des programmes

Cette exigence de discipline est très réelle et on peut dire qu'elle motive tout le mouvement de professionnalisation de la programmation. Si le manque de rigueur de la programmation amateur que nous avons illustré tout à l'heure[34] ne prêtait certainement pas à conséquence étant donné la simplicité du problème et la faiblesse de son enjeu, il devient rapidement ingérable dans de grands projets qui mobilisent des dizaines, voire des centaines de programmeurs. Les problèmes qui se posent deviennent également qualitativement différents au-delà d'une certaine taille. Par exemple des problématiques d'architecture logicielle* surviennent concernant l'organisation générale des milliers de modules de programme et leur communication les uns avec les autres. Des problématiques de gestion d'équipe, de calendrier, d'évolution de besoins apparaissent également. Ce changement qualitatif a été très tôt identifié par les programmeurs, qui on appellé cela *programmer-en-grand* (*programming-in-the-large*), par opposition à la programmation qui ne compte que quelques dizaines de modules, et qui fait intervenir au plus quelques programmeurs[35].

Au-delà de ces problèmes d'architecture et de coordination, un très grand programme est le plus souvent amené à devoir s'adapter à des environnements d'usage très différents. Un logiciel de facturation, par exemple, ne fonctionne jamais seul, il est toujours intégré à un logiciel de prise de commande, un autre de comptabilité et à des dizaines d'autres modules logiciels qui s'interfacent avec lui quotidiennement. Il doit pouvoir s'adapter à une grande diversité de facteurs d'environnement, qui comprennent le matériel informatique lui-même, les logiciels d'interface, les langues des usagers, les contraintes réglementaires, les besoins spécifiques des entreprises. Il peut y avoir des pannes, des erreurs, des fichiers corrompus etc. Il y a donc une exigence de *robustesse* du logiciel face à la combinatoire immense des situations dans lesquelles il peut être plongé.

Enfin il est probable qu'on attende d'un programme aussi coûteux à réaliser qu'il soit durable – certains sont encore en activité cinquante ans après leur premier développement – et cela implique souvent qu'il puisse être adapté à des conditions

34. Voir plus haut, § 28.
35. Cette expression fut proposée initialement par (DeRemer et Kron 1975). Nous y revenons à la section suivante.

d'usage et à des besoins nouveaux, à un coût non prohibitif.

Cette dernière exigence de durabilité, combinée à celle de robustesse, est le fil conducteur de la discipline qu'on requiert des programmeurs : il s'agit de lutter contre l'obsolescence naturelle à laquelle tout programme semble condamné, avec un rythme qui croît avec sa taille. Une des problématiques économiques essentielles de l'informatique n'est donc pas l'innovation, mais bien la maintenance des systèmes en place, et donc de leurs programmes. Comme le dit également la préface de l'ouvrage *Coder Proprement* :

> Même dans l'industrie automobile, la majeure partie du travail ne réside pas dans la production mais dans la maintenance – ou son évitement. En programmation, 80% ou plus de ce que nous faisons est étrangement appelé « maintenance » : l'acte de réparer. Plutôt qu'adopter la focalisation occidentale typique sur la *production* de bons logiciels, nous devrions nous voir davantage comme des réparateurs à domicile ou des mécaniciens automobiles[36].

Les conséquences de la bonne ou mauvaise maintenance d'un système peuvent être majeures. L'auteur, R.C. Martin, donne l'exemple d'une entreprise de logiciels qui, après le succès initial de son produit, connut des difficultés à le faire évoluer. Les erreurs se multipliaient d'une nouvelle version à une autre, si bien que ses utilisateurs finirent pas abandonner l'outil et l'entreprise fit faillite. Un des anciens employés raconta à Martin, des années après, ce qui s'était passé :

> Ils avaient mis le produit sur le marché trop précipitamment et avaient de ce fait laissé un grand désordre dans le code. Au fur et à mesure qu'ils ajoutaient plus de fonctionnalités, le code allait de pire en pire jusqu'à devenir complètement ingérable. *C'était le mauvais code qui avait fait chuter l'entreprise*[37].

Tout logiciel a ainsi un cycle de vie et une sorte d'obsolescence naturelle contre laquelle il faut lutter. Cette obsolescence a deux causes, l'une externe, et l'autre interne.

L'obsolescence externe est, d'une certaine manière, en proportion directe de l'innovation des technologies qu'elle utilise : les mises à jour successives des systèmes d'exploitation, des périphériques, des compilateurs, etc. entraînent la sclérose rapide de tout logiciel qui ne s'y adapte pas, puisque les développeurs ne disposent plus des outils nécessaires pour le modifier. Ainsi, dans l'expérience personnelle de l'auteur, un logiciel de facturation, développé par les équipes internes de l'entreprise, devint subitement obsolescent par la seule décision d'un grand

36. (R. C. Martin 2009, p. xx).
37. (ibid., p. 3).

éditeur de ne plus maintenir la brique logicielle centrale sur laquelle l'application reposait. Cet éditeur proposait à l'entreprise de *migrer* celle-ci vers une nouvelle plateforme, pour un coût prohibitif. Cette dépendance envers des composants obsolètes est appelée communément *dette technologique* car, s'il n'y a pas nécessairement urgence à les remplacer, il y a la certitude qu'ils devront l'être un jour, pour un coût qui augmente en général avec le temps.

Dans l'exemple ci-dessus, Martin parle cependant d'une seconde raison d'obsolescence, cette fois interne, et qui est capturée dans l'idée de *désordre* (*mess*) qui revient sans cesse sous sa plume. Ici, c'est l'image de l'entropie qui vient à l'esprit, qui rend un système de plus en plus difficile à transformer, et que chaque transformation augmente à son tour. Pour pallier des défauts de conception, on doit « bricoler » des solutions qui, n'ayant pas de logique systématique, deviennent autant de facteurs de fragilité pour les développements futurs. Le code leur correspondant est d'abord difficile à comprendre, puisqu'il ne découle pas d'un principe simple : si on ne connaît pas l'intention qui l'a motivé, comme la gestion d'une exception, il est pratiquement impossible de comprendre à quoi il sert. Comme ces bricolages sont souvent faits dans l'urgence, ils ne sont pas toujours documentés. Un informaticien raconte par exemple comment, quinze ans après avoir développé un petit programme pour un opérateur télécoms alors qu'il en était le sous-traitant, il s'aperçut au cours d'une discussion que celui-ci était toujours actif. Bien qu'il ralentît sérieusement la performance de certains traitements de données chez cet opérateur, les informaticiens n'osaient pas le désactiver, car ils ne savaient pas quelle était sa fonction[38]. Il semble qu'un logiciel, au fil et à mesure de son évolution, s'alourdit de ses ajouts et modifications successives, jusqu'à s'affaisser sur lui-même.

Cette seconde forme d'obsolescence, qui est la raison essentielle de ce qu'on nomme *legacy software* en anglais, et en français *dette informatique* en général, est en quelque sorte un scandale pour la raison. Elle est en tous les cas vécue comme telle par les clients ou les usagers des systèmes informatiques. Ils ne comprennent pourquoi il est si difficile de faire des modifications au système, aussi petites soient-elles, et surtout pourquoi elles prennent autant de temps et coûtent autant d'argent. Les informaticiens eux-mêmes la vivent très mal, du fait d'abord de la réputation d'incompétence qu'on leur fait, mais aussi et surtout de la pénibilité et de l'ennui de la tâche. Cela est très bien décrit par Martin :

38. François Ruisz, communication personnelle.

> Avez-vous déjà été considérablement embarrassé par du mauvais code ? Si vous êtes un programmeur expérimenté, vous avez certainement ressenti cet embarras à plusieurs reprises. Et de fait, nous avons un nom pour cela. Nous l'appelons *patauger*. Nous pataugeons dans le mauvais code. Nous rampons dans un bourbier de ronces enchevêtrées et de pièges cachés. Nous luttons pour trouver notre chemin, espérant une piste, un indice de ce qui se passe ; mais tout ce que nous voyons est toujours plus de code insensé.
>
> Bien sûr, vous avez déjà été embarrassé par du mauvais code. Mais alors, pourquoi l'avez-vous écrit [39] ?

L'aspect scandaleux de cet état de choses est sensible dans la question rhétorique finale de Martin : car contrairement à la dette technologique, le mauvais code est en général le résultat de l'activité même de programmer, de l'équipe même qui en souffre. Certainement celle-ci a dû répondre à des pressions et des contraintes externes et, sur le long terme, le changement des personnes implique une perte de connaissance collective. Cependant, on peut imaginer qu'il est toujours possible de s'organiser face à ces contraintes, et surtout, de « pouvoir recommencer » si nécessaire sur de bonnes bases. De la même manière que les ingénieurs conçoivent régulièrement de nouvelles machines en y intégrant les enseignements des difficultés rencontrées lors de la maintenance des précédentes, il semble que le mieux à faire, devant un logiciel obsolète, est de le redévelopper à zéro.

Cette idée semble d'autant plus légitime que le développement d'un nouveau logiciel paraît plus simple que celle d'une nouvelle machine : il n'y a pas de pièces matérielles à acheter, voire à faire fabriquer sur mesure, pas d'usinage ou de montage mécanique, de tests matériels, etc. Le programmeur semble, comme l'exprimait Brooks dans un texte déjà cité[40], « ne travailler qu'à une distance minimale de la pure pensée, construire des châteaux dans les airs ». Comment donc se fait-il que les programmeurs ne redémarrent pas à zéro quand un logiciel présente des signes d'obsolescence ?

Martin décrit de manière saisissante la difficulté opérationnelle à réaliser cette idée. L'histoire est racontée sous le mode ironique, mais nous-même avons eu l'expérience personnelle exacte de l'état de choses qu'il décrit :

> Finalement, l'équipe se révolte. Elle informe la direction qu'elle ne peut pas continuer à développer dans cette base de code odieuse. Elle réclame une refonte. La direction ne veut pas consacrer les ressources à une refonte totale du projet, mais

39. (R. C. Martin 2009, p. 3).
40. Voir plus haut, § 10.

> elle ne peut nier que la productivité est lamentable. Finalement, elle se plie aux exigences des développeurs et autorise la grande refonte du siècle (*the grand redesign in the sky*).
>
> Une équipe spéciale (*tiger team*[41]) est sélectionnée. Tout le monde veut faire partie de cette équipe parce que c'est un projet greenfield*, on peut tout y recommencer et créer quelque chose de vraiment beau. Mais seuls les meilleurs et les plus brillants sont choisis pour en faire partie. Tous les autres doivent continuer à maintenir le système actuel.
>
> Maintenant, les deux équipes sont dans une course de vitesse. L'équipe spéciale doit construire un nouveau système qui fait tout ce que fait l'ancien, et de plus, suivre les changements qui continuent d'être apportés à celui-ci, [car c'est la condition que la direction a posé au changement de système].
>
> Cette course peut durer très longtemps. J'ai vu des projets où cela prenait dix ans. Et au moment où elle s'achève, les membres originaux de l'équipe spéciale sont partis depuis longtemps, et les membres actuels exigent que le nouveau système soit conçu à nouveau parce qu'on y trouve un tel désordre[42].

Les raisons de la grande difficulté à *remplacer* un système informatique sont mystérieuses, et sont selon nous un marqueur de la spécificité épistémique de la programmation. En première approche, le problème que rencontrent les informaticiens est qu'un système ancien incorpore des centaines, voire des milliers, de règles métier* ajoutées au fil des années de manière incrémentale et désordonnée, si bien qu'embrasser d'un seul coup d'œil tout leur spectre est un défi en soi. Il faut les identifier, les documenter, les trier, les intégrer à la nouvelle architecture, négocier avec les utilisateurs, etc. En d'autres termes, le problème de fond n'est pas vraiment technique (savoir comment faire quelque chose), mais plutôt de visée (savoir que faire). C'est apparemment cette imbrication inséparable de la spécification et de la réalisation qui semble rendre la programmation paradoxalement moins agile que les autres disciplines d'ingénierie plus classiques. Nous y reviendrons à la fin de notre réflexion[43].

Face au risque d'obsolescence, et au « miroir aux alouettes » que constitue l'idée du redéveloppement « à zéro », les programmeurs ont compris que la voie de loin la plus préférable était de *prévenir* ce risque dès le départ, en s'imposant des disciplines quotidiennes dans le développement et la maintenance de programmes.

41. Il s'agit d'une expression anglo-saxonne utilisée dans le monde informatique pour dénommer une équipe d'intervention de haut niveau visant à résoudre des problèmes critiques.
42. (R. C. Martin 2009, p. 5).
43. Voir plus loin § 233.

De nombreux principes reflètent cet état d'esprit que la profession cherche à embrasser. Un des premiers commandements faits au jeune programmeur est : « *Ne fais aucun mal !* » (*Do No Harm !*) – bien entendu l'objet de cette prudence est le système informatique qu'il va être amené à modifier[44]. Un autre commandement populaire est le « principe du boy-scout » : tout programmeur doit laisser un programme qu'il a modifié *en meilleur état* qu'il ne l'a trouvé[45].

§ 31. La revue de code et ses raisons

Une bonne illustration de ces principes se trouve dans la *revue de code* (*code review*), une pratique courante d'assurance-qualité dans le développement logiciel. Elle consiste à soumettre, à intervalles réguliers, le code écrit par un programmeur à la critique d'un de ses collègues afin que celui-ci puisse y détecter des erreurs potentielles, des points d'amélioration, et surtout s'assurer de sa compréhensibilité[46].

On peut s'étonner doublement d'une telle pratique. D'une part, comme on l'a vu, l'écriture de programme est très itérative : on se relit donc constamment. Se relire est une activité naturelle quand on écrit, qu'il s'agisse de la rédaction d'un rapport, de notes de réflexion ou de l'écriture d'un programme. D'autre part, on pourrait penser que l'*exigence* de relecture est moins importante pour un programmeur que pour tout autre écrivain, puisqu'il peut bénéficier d'un retour immédiat et objectif sur les effets de sa production. L'exécution de son programme lui montre s'il a atteint sa visée ou non, tandis que l'écrivain doit en général faire l'effort de se relire « avec du recul » pour sentir les effets potentiel de son texte sur son destinataire. Au contraire, dans l'expérience de programmation que nous avons relaté, les tentatives échouées d'exécution du programme ont constamment piloté sa relecture et sa modification.

C'est donc à un autre type de lecture qu'invite la revue de code, dont on semble penser que le programmeur-auteur est incapable, puisqu'on la confie à un tiers. On semble se méfier, justement, des relectures « à chaud » comme celles que nous avons faites, téléguidées par le débogage*. La revue de code vise à mettre en place des « garde-fous » complémentaires, qui sont au moins au nombre de trois.

D'abord, on ne peut tester en général toutes les situations possibles dans les-

44. (Jacobson et al. 2019, p. 10).
45. (R. C. Martin 2009, p. 14).
46. (Greiler 2020).

quelles le programme sera exécuté, comme par exemple toutes les combinaisons de paramètres qui lui seront fournis. Une bonne revue de code comprend également l'inspection des cas de tests que le programmeur déclare avoir réalisé. (R. C. Martin 2009, p. 268) montre l'exemple d'une telle revue qui commence par la redéfinition complète des cas de tests du programme en question. Mais surtout, le programmeur tiers, visitant le code avec un œil non prévenu, et avec la connaissance d'autres parties du logiciel, a plus de chance d'être sensible à des failles logiques du raisonnement conduisant à l'oubli de certaines classes de situations possibles.

Ensuite, un programme peut fonctionner parfaitement mais avoir des « effets de bords » (*side-effects*) sur d'autres programmes avec lesquels il interagit. Un premier exemple est donné par R. C. Martin (ibid., p. 44). Il s'agit d'une fonction d'authentification, qui vérifie que le mot de passe d'un utilisateur est correct. Comme l'authentification advient en général lors de l'accès à une application, le programmeur avait inclus dans son code la fonction d'initialisation de la session. Cela est une erreur grave, car si cette fonction est appelée à nouveau *au cours* de la session (par exemple pour valider une action importante, comme un virement bancaire), cette session se trouvera réinitialisée. Le principe qu'en tire Martin est qu'une fonction ne doit faire *qu'une seule chose* et être explicite à ce propos.

Un second exemple concerne des ressources partagées par plusieurs programmes. L'exemple classique est celui d'une chambre d'hôtel qui peut être réservée au même moment par deux agents de voyage différents, car elle leur apparaît à tous deux comme libre[47]. De ce fait, des outils pour donner l'exclusivité temporaire ont été mis en place dans les langages de programmation, comme les sémaphores[*]. Mais là encore, des effets de bord peuvent également apparaître. Ainsi, une application qui doit mettre à jour des fichiers peut oublier de les refermer – ce qui ne lui pose aucun problème, mais bloque leur accès à toute autre application[48]. Il ne s'agit là que de quelques exemples : toutes les « vulnérabilités » informatiques qui permettent le piratage proviennent, par définition, de l'exploitation d'effets de bord d'applications mal conçues.

Enfin, lorsqu'on doit maintenir et modifier un programme sur la longue durée, on est confronté à un tout autre type d'expérience de relecture du code, celle que Robert Martin appelle « patauger », dans la citation ci-dessus. Cette relec-

47. (D. Thomas et Hunt [1999] 2020, p. 174).
48. (ibid., p. 118).

ture « à froid » de vieux code, qu'on n'a souvent pas écrit soi-même, est pourtant l'expérience la plus commune du programmeur :

> Le ratio de temps passé à lire plutôt qu'à écrire est bien au-dessus de 10:1. Nous sommes *constamment* en train de lire du vieux code, dans l'effort d'en écrire du nouveau. Parce que ce ratio est si grand, nous voulons que cette lecture soit aisée, même si cela en rend l'écriture plus difficile[49].

Il y a ici une expérience élémentaire : dans le feu de l'écriture d'un programme, sa logique interne semble évidente au programmeur, comme la séquence de ses enchaînements, le rôle des différentes variables, l'emplacement de telle fonction. Toute cette évidence disparaît quelques jours ou quelques semaines plus tard, et on peut se retrouver, si on n'a pas suivi des règles élémentaires d'écriture, face à un charabia incompréhensible. C'est d'abord ce risque que la revue de code cherche à prévenir, en s'assurant que le programme est compréhensible *à froid*, par quelqu'un qui n'a pas participé à son écriture. Par exemple, il s'agit de s'assurer que le nom des variables est assez explicite pour décrire leur contenu ou leur fonction.

D'autres mauvaises pratiques plus subtiles peuvent également être détectées. Nous nous contentons d'illustrer une des plus connues, la duplication de code au sein d'un même programme. Si ce code doit être modifié, il devra l'être à tous les endroits, avec les risques d'oubli et d'incohérence que cela entraîne. R. C. Martin (ibid., p. 38) donne l'exemple des branchements d'exécution par distinction de cas, comme ici :

Code 2.5 – Programme Java calculant la paie en fonction de la situation de l'employé

```
1     public Money calculatePay(Employee e)
2     throws InvalidEmployeeType {
3         switch (e.type) {
4             case COMMISSIONED:
5             return calculateCommissionedPay(e);
6             case HOURLY:
7             return calculateHourlyPay(e);
8             case SALARIED:
9             return calculateSalariedPay(e);
10            default:
11            throw new InvalidEmployeeType(e.type);
12        }
13    }
```

49. (R. C. Martin 2009, p. 14).

Cette fonction consiste simplement à appeler différentes fonctions de calculs de la paie (lignes 5-7-9) en fonction du statut de l'employé (lignes 4-6-8, instruction `case`), renseigné dans la variable `e.type` (ligne 3), et à renvoyer un message d'erreur si sa valeur n'est pas reconnue (ligne 2-10-11). Le problème d'une telle manière de faire est que d'autres fonctions, comme celle consistant à savoir quel est le jour de la paie pour chaque employé, vont devoir adopter la même structure de branchement. Ainsi, si de nouveaux types d'employés sont ajoutés plus tard, toutes ces fonctions devront être mises à jour. Cela est un facteur important de risque car, sur un programme constitué de milliers de lignes, il peut être difficile de repérer de telles structures.

Tous ces exemples mettent en évidence le caractère systémique de tout programme conséquent. Le principe *« Ne fais aucun mal ! »* est pertinent car un programmeur ne doit pas seulement assurer l'effectivité de son propre code, mais surtout, ce qui est bien plus difficile, prévoir toutes les situations possibles et éviter les effets systémiques pernicieux. Il doit également s'assurer que son code reste « ouvert » à des modifications futures afin qu'elles n'aient pas à « patauger », comme le dit Martin.

La revue de code n'est cependant qu'une pratique parmi d'autres que les programmeurs professionnels ont mis en place pour assurer la robustesse et la durabilité des grands programmes. Dans la section suivante, nous allons tenter d'en donner un aperçu plus général.

2.3 La fragmentation des méthodes professionnelles

Il s'agit dans cette section de donner un aperçu de la programmation professionnelle, couramment appelée *ingénierie logicielle* (*software engineering*) aujourd'hui. En § 32, nous brossons une courte histoire des méthodes qui ont marqué l'émergence de cette discipline et tentons d'identifier les éléments importants de ce qui constitue l'« état de l'art » aujourd'hui. Dans le développement suivant § 33 nous illustrons ce qu'est la programmation professionnelle par un petit exemple.

Cependant, l'idée d'une industrialisation possible de la programmation a été régulièrement contestée au cours de l'histoire de l'informatique, au motif que chaque projet requérait une résolution de problème spécifique difficilement compatible avec la division du travail et l'anonymisation des activités. Dans cette optique, c'est plutôt le statut d'artisan ou d'auteur (*craftsmanship, authorship*) du

programmeur qui est mis en avant (§ 34). Nous étudions enfin une tentative récente de synthèse entre ces deux tendances opposées, qui aboutit selon nous à des paradoxes qui rendent sensible l'impossibilité d'une caractérisation axiologique de la programmation (§ 35).

§ 32. La programmation-en-grand

Nous présentons d'abord une très rapide esquisse de l'histoire des méthodes de développement logiciel, en suivant Jacobson et al. (2019, p. 21, 341).

La notion de *programmer-en-grand* paraît dans le contexte de ce qu'on appelle *la crise du logiciel*, dont le constat est dressé par deux conférences célèbres de l'OTAN à la fin de l'année 1969, et qui motivent (entre autres) la notion d'*ingénierie logicielle* (*software engineering*)[50]. Il s'agit d'importer en informatique la rigueur des méthodes de conception qu'on trouve dans l'ingénierie. Le programme est alors assimilé, analogiquement ou au sens propre, à un artefact physique ou à un produit.

Ce sont ces conférences qui ont donné sa légitimité à la première méthode conséquente de développement logiciel, la programmation structurée*, qui insiste sur la nécessité d'organiser hiérarchiquement la programmation en unités modulaires. Cette méthode était centrée sur la notion de fonction à entrée - sortie, autour de laquelle l'analyse modulaire et hiérarchique du problème à résoudre était organisée. Un des problèmes de ces méthodes est qu'elles laissent irrésolue la question de la gestion des données partagées entre plusieurs fonctions : où localiser logiquement celles-ci, comment établir des règles d'accès ou de mise à jour, etc.

À peu près en parallèle, une première méthode de travail collectif est formalisée, qu'on appelle le modèle en cascade* ou encore modèle de développement en V*, prescrivant d'organiser un projet de programmation en phases successives comme notamment l'expression de besoins, la conception, l'implémentation et la vérification (voir figure 2.1). Cette séquence, on l'a vu, représente plutôt une reconstruction logique du processus de programmation que sa réalité, où des itérations sont toujours nécessaires. Cependant, elle correspond aussi à une division naturelle du travail, par exemple entre maître d'ouvrage, architecte, codeur, et utilisateur pour reprendre les quatre phases illustrées en figure 2.1. À ce titre, *le passage de témoin* d'une équipe à l'autre est un lieu de négociations et potentiellement de frictions, d'autant plus que la tendance à la sous-traitance du codage, qui

50. On peut se reporter ici par exemple à la relation qu'en donne (MacKenzie 2004, p. 34).

se développe à cette époque, implique une contractualisation nette des spécifications. Ces négociations ralentissent le projet puisqu'elles engagent la responsabilité de chaque partie. Dans les projets complexes et tendus, les attitudes se crispent, puisque ni les programmeurs ni les clients n'ont plus le droit de remettre en cause des spécifications une fois cette validation obtenue ; quant aux responsables des spécifications, situés à l'interface entre ces deux groupes, ils sont sur la défensive[51].

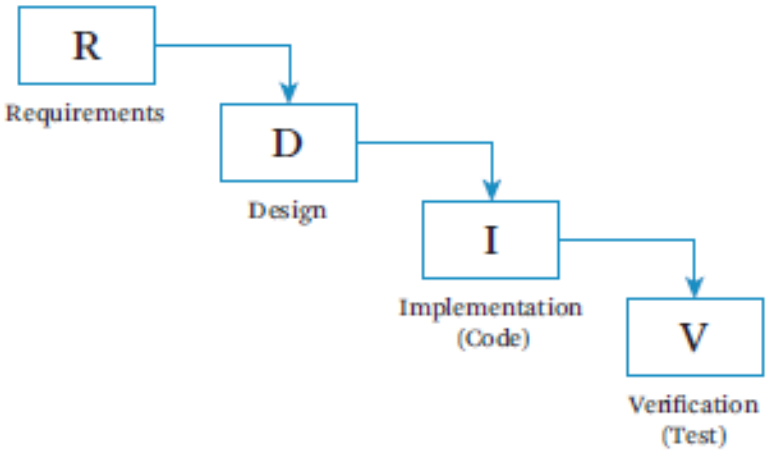


FIGURE 2.1 – La méthode de développement en cascade
(Source : (Jacobson, 2019, p. 20))

Dans les années 1990 de nouvelles méthodes se fondent sur l'idée de compartimentation du logiciel en petites unités complètement autonomes, encapsulant à la fois les fonctions et les données en des ensembles cohérents, et communiquant entre elles par des interfaces explicites. En partie inspirées et associées à la programmation objet*, elles convergent dans le modèle UML[52], qui devint le standard dominant du développement logiciel au début des années 2000, soutenu par une association industrielle de standardisation, l'Object Management Group. Privilégiant des outils diagrammatiques visuels pour expliciter les spécifications, et néanmoins formalisables dans un langage de spécification précis, une large panoplie de langages, d'outils, de formations, de certifications en font, un temps,

51. (JACOBSON et al. 2019, p. 20). Nous avons nous-même observé cet état de fait lors de larges projets informatiques.
52. (BOOCH, JACOBSON et RUMBAUGH 2005).

une *lingua franca* de la profession. L'idée générale de ces approches est de rendre la programmation *vraiment* modulaire, en donnant le maximum d'autonomie à chaque équipe responsable d'un module, en échange d'une adhérence stricte à l'architecture et la spécification des interfaces. Les approches SOA* et micro-services* peuvent être vues comme des reformulations de cette philosophie[53].

À partir des années 2000, les méthodes agiles* se focalisent sur les dimensions humaine et temporelle de la programmation. Un projet ayant des contraintes lourdes est lent, ce qui démotive les programmeurs et fait fuir les talents. Au contraire, un projet qui itère rapidement sur lui-même permet de libérer les contraintes, puisqu'on peut intégrer à l'itération suivante de nouvelles demandes ou des modifications. Il motive les programmeurs, ce qui les rend également plus engagés dans le projet. Cette approche repose sur plusieurs principes, comme le dialogue constant entre clients et programmeurs, l'acceptation du changement des spécifications et la fixation de sprints* de programmation à durée limitée[54]. Elle requiert, techniquement, de savoir mettre en production* régulièrement et fréquemment de nouvelles versions du logiciel, ce qui donne lieu, à la limite, à la notion d'intégration continue*.

Le résultat cumulé de ces évolutions se matérialise dans la structure de l'environnement de travail de la programmation, qui s'est fortement technicisé, et qui n'a plus grand chose à voir avec la programmation amateur :

1. L'architecture logicielle est devenue la fonction nodale de la programmation, le lieu où se prennent toutes les décisions majeures de conception. Une mauvaise architecture, on l'a vu, peut rendre le code difficile à maintenir et peu évolutif.
2. Tout projet logiciel implique également le choix (parfois fortement contraint) d'un ensemble cohérent de technologies. La pile technologique*, comme on l'appelle, ne se résume pas au choix du langage de programmation et du système d'exploitation*, mais porte également sur les technologies de base de données, de serveurs web, de poste de travail, etc. L'essentiel est que ces technologies soient aisément interopérables. Microsoft .NET est ainsi une pile technologique* prête à l'emploi, et d'autres pile technologiques* sont de plus en plus souvent pré-assemblées dans l'informatique en nuages*, comme par exemple chez AWS (Amazon Web Services). La conteneurisation* consiste à

53. (Jacobson et al. 2019, p. 24).
54. (Beck et al. 2001).

assembler en une seule brique logicielle l'application développée ainsi que toute sa pile technologique* sous-jacente, de telle sorte qu'elle puisse être déployée aisément sur n'importe quelle infrastructure.

3. R. C. Martin (2009) a lancé le mouvement du *code propre*. Plutôt que s'engager dans des grandes refontes de logiciel, ce mouvement incite les programmeurs à appliquer des principes « hygiéniques » de développement dans n'importe quelle tâche, et s'inspire de la philosophie industrielle japonaise d'amélioration continue. Elle se traduit concrètement par un ensemble de techniques d'analyse et de modification de code, appelées réusinage*.
4. L'outillage s'est perfectionné, par exemple quant à la gestion des tests, qui peuvent être en grande partie automatisés, ou la gestion des versions d'un logiciel. Sur ce sujet, le standard actuel est le logiciel Git, qui facilite la réversibilité de toute évolution et surtout permet la traçabilité et la documentation des changements ; il donne de l'importance aux revues de code. Il existe également des outils d'automatisation de l'audit et du réusinage* de code.
5. Les langages de programmation eux-mêmes et leurs compilateurs se sont adaptés pour répondre à ces nouveaux environnements et aux besoins de maintenance du code. Ils sont devenus plus robustes et incorporent parfois nativement des méthodes de vérification qui devaient être effectuées par le programmeur auparavant[55].
6. La gestion de projet informatique s'est enfin dotée et de « bonnes pratiques » concernant le travail collaboratif, la gestion des personnes, la mesure du progrès, la sous-traitance, etc. Elle s'inscrit également dans une tendance d'évolution des organisations informatiques où, par exemple, la séparation historique entre développement et opérations est remise en cause au profit d'équipes spécialisées par domaine applicatif, responsables de leur logiciel tout au long de son cycle de vie (Devops*).

§ 33. Exemple d'une application professionnelle

Afin de donner un aperçu, même minime, du contenu de cette programmation professionnelle, nous avons eu accès au projet de développement logiciel d'une grande société de services informatiques. L'application en question était développée pour répondre à ses propres besoins internes. Il s'agissait de permettre aux salariés d'accéder aux différents services offerts par les campus de l'entreprise

55. Nous en donnons un exemple avec le langage Eiffel, en § 56.

(poste de travail, restaurant, salles de réunions, etc.). La partie qui nous intéresse concernait la mise à disposition des salariés d'un outil de réservation de salles de réunion. Au cours de son déploiement mondial (l'entreprise est présente partout dans le monde, et compte plus de 100 000 salariés) cet outil se révéla inadapté pour les campus indiens, gigantesques, dont certains comportaient plus d'une centaine de salles. L'application se révéla lente et inutilisable sur ces sites. Les utilisateurs demandèrent donc une amélioration de l'outil afin qu'ils puissent consulter leurs options étage par étage plutôt que dans le désordre. Les captures d'écran 2.2 montrent le résultat attendu.

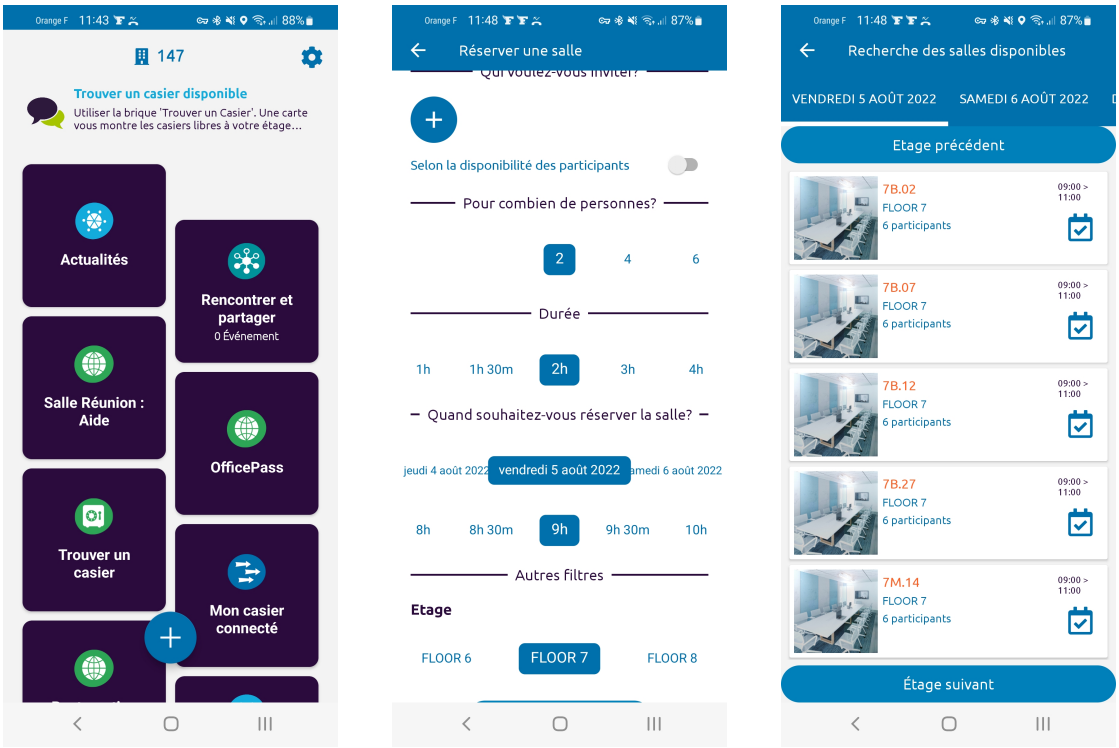


FIGURE 2.2 – Captures d'écran d'une application mobile de réservation de salles,organisée par étages
Source : Cap Gemini

Le processus de développement a suivi les étapes suivantes. D'abord une « demande utilisateur » a été officiellement formulée, afin qu'elle rentre dans la file d'attente des demandes concernant cette application. Celles-ci sont gérées dans JIRA, un logiciel de gestion de projet agile* très populaire en ce moment (début 2023). Il permet de gérer les priorités des demandes, de les organiser en unités de

développement courtes (scrums*), et de suivre leur progrès selon une procédure définie par l'entreprise. Il permet également de documenter ces besoins, de les décrire en fonctionnalités et en scénarios d'usage. Dans ce cas-ci, elle était formulée en langage courant de la manière suivante :

> Il existe un paramètre dans le service « *Réserver une salle de réunion* » qui permet de choisir l'étage souhaité. À la fin de la liste des salles de réunion proposées, il y a un bouton « *Afficher l'étage suivant* ». Lorsque l'utilisateur clique sur le bouton, les salles de réunion actuellement proposées disparaissent et l'étage suivant est affiché. Un bouton avec « *Afficher l'étage précédent* » apparaît au-dessus de la liste.

La demande est assez informelle et courte ; il y a pas mal d'ellipses dans la description du parcours de l'utilisateur, car ce qui reste inchangé est passé sous silence (par exemple quels sont les autres critères pour choisir une salle de réunion, quelles sont les salles affichées en premier, etc.)

Après validation, la demande fut soumise à un architecte qui étudia la manière dont cette demande devait être traitée. Comme il s'agit d'une demande simple, il se contenta de la séparer en deux parties, front-end* et back-end*. La première demande concerne l'ajout des boutons demandés sur l'interface utilisateur, la seconde la modification de la fonctionnalité existante.

La première demande fut décrite comme suit :

> Description :
> *« En tant qu'utilisateur, je souhaite pouvoir voir la salle de réunion de l'étage sélectionné lorsque je clique sur rechercher. »*
> Parcours :
>
> 1. Après avoir cliqué sur le bouton de recherche, le service affiche toutes les salles de réunion disponibles, tel que cela se fait actuellement, en fonction du créneau horaire et UNIQUEMENT à l'étage sélectionné.
> 2. À la fin de la liste des salles de réunion proposées, il y a un bouton « *Afficher l'étage suivant* ». Lorsque l'utilisateur clique sur le bouton, les salles de réunion actuellement proposées disparaissent et le service affiche les salles de réunion disponibles à l'étage suivant.
> 3. Un bouton avec « *Afficher l'étage précédent* » apparaît au-dessus de la liste. Lorsque l'utilisateur clique sur le bouton, les salles de réunion actuellement proposées disparaissent et le service affiche les salles de réunion de l'étage précédent.
> 4. Lorsque l'utilisateur est au premier étage (il n'y a plus d'étage précédent) le bouton « *Étage précédent* » disparaît.

5. Lorsque l'utilisateur est au dernier étage (il n'y a plus d'étage suivant) le bouton « *Étage suivant"* » disparaît.

Cette spécification ne dit pas grand chose de plus que la demande initiale. Elle est seulement débarrassée de quelques ambiguïtés et les exceptions sont explicitées (items 4 et 5 du parcours). Ce texte est accompagné des visuels de la figure 2.2, qui montrent ce à quoi le résultat doit ressembler.

La seconde demande (back-end*) concerne la refonte de la fonction de sélection des étages disponibles. La spécification fut la suivante :

> Description :
> Dans l'API*, ajouter la possibilité de choisir l'étage : Fournir un nouveau paramètre `floorID` permettant de spécifier quel étage l'utilisateur demande.

Cette spécification est très succincte car il s'agit ici simplement d'enrichir une spécification déjà existante. Cependant, on va le voir, le manque de détail va amener le programmeur à improviser une spécification complémentaire. Enfin ces demandes furent transformées en code. Le langage utilisé ici est Swift (conçu pour l'iPhone), et la plateforme technologique sous-jacente est Microsoft .NET. Le code 2.6 correspond à la fonction qui réagit au clic du bouton « *Étage suivant* ».

CODE 2.6 – Fonction renvoyant les salles disponible selon l'étage sélectionné

```
1  func loadMoreResults(sender: moreResultsCell) {
2      guard let currentFloorIdString = self.bookingParameters?.floorId
           else {
3          return
4      }
5      let currentFloorId = String(currentFloorIdString) // retrieve
           floorId
6      guard let floors = Global.shared.building?.floors else {
7          return
8      }
9      let currentIndex = floors.firstIndex{$0._id == currentFloorId}
10     if let nextFloorId = floors[currentIndex! + 1]._id {
11         if let floorPois = Global.shared.poisByFloor[nextFloorId] {
12             self.roomsToSearch = floorPois.filter({$0.type?.name ==
                   roomConferenceType})
13             self.bookingParameters?.floorId = Int(nextFloorId)!
14         }
15     } else {
16         if let floorPois = Global.shared.poisByFloor[floors[0]._id!]
               {
```

```
17              self.roomsToSearch = floorPois.filter({$0.type?.name ==
                    roomConferenceType})
18              self.bookingParameters?.floorId = Int(floors[0]._id!)!
19          }
20      } // go back to first
21      self.retrieveDayResult()
22  }
```

La première chose qui frappe lorsqu'on lit le code, c'est la longueur des noms des variables : ils sont très explicites sur leur contenu et leur rôle, afin de faciliter leur compréhension. La lecture est par contre rendue plus difficile par la présence de variables globales*, dont il faut deviner les propriétés et les fonctions. Celles-ci sont `Global.shared.building?.floors` qui désigne la liste des étages de l'immeuble sélectionné, et `Global.shared.poisByFloor` la liste des salles par étage. Par ailleurs, cette fonction est dépendante d'un objet*, la requête en cours, dont les propriétés et procédures sont introduites par le préfixe `self.`

Nous n'allons pas expliciter ce que fait cette fonction ligne à ligne. Son rôle principal est de fournir à la requête en cours la liste des salles de réunion sur lesquelles réaliser la recherche. Cette liste est enregistrée dans la propriété de la requête nommée `self.roomsToSearch` (ligne 12 ou 17, selon les cas). Par ailleurs, le paramètre de la requête concernant l'étage demandé `self.bookingParameters?.floorId` est mis à jour à la valeur de l'étage suivant (ligne 13 ou 18) – une action qui était restée implicite dans la spécification.

Une fois que cela est fait, la fonction se termine (ligne 21) en appelant la fonction de recherche proprement dite, qui va sélectionner parmi cette liste les salles disponibles, correspondant aux critères de l'utilisateur.

La fonction est organisée en deux parties. La première, lignes 2-9, déclare des variables utiles pour la suite. La seconde (lignes 10-20) est organisée autour d'un branchement ligne 10 : si on trouve un « *étage suivant* », alors la fonction renvoie la liste complète des salles de cet étage. Sinon (ligne 15), elle renvoie la liste des salles de réunion du premier étage de l'immeuble : cela est d'ailleurs explicité par un commentaire ligne 20.

On peut s'étonner de cette condition, puisqu'elle semble contredire la spécification : il ne devrait pas être possible de sélectionner un étage suivant si on consulte déjà les salles du dernier étage, puisque le bouton « *étage suivant* » ne devrait pas apparaître. On voit qu'ici le développeur du back-end*, bien qu'il soit certainement averti de cela, a choisi de traiter le cas où cela *se produirait tout de même*.

Par principe, il se méfie du front-end*, qui peut s'être trompé ou avoir changé d'avis. Deux options s'ouvrent alors à lui : renvoyer une exception* ou programmer un comportement par défaut. C'est ce qu'il choisit ici de faire, en renvoyant la liste des salles du réunion du premier étage de l'immeuble.

Ce choix illustre un principe important de la programmation professionnelle : découper le développement en zones de responsabilité claires (ici front-end* et back-end*) et demander à chacune d'adopter un principe de précaution concernant les conditions dépendantes de zones tierces. C'est cela qui permet à chaque zone d'être maintenue et d'évoluer de manière indépendante sans avoir d'effets de bord sur les autres.

D'autres enseignements peuvent être tirés de ce court exemple, notamment par contraste avec l'exemple amateur du développement § 28.

Tout d'abord, le processus apparaît bien moins itératif. Il semble progresser linéairement à travers des étapes clairement définies. Cela n'est cependant qu'une apparence. Au sein de chaque étape – l'expression de besoins, le choix d'architectures, les spécifications, le codage – de nombreuses tentatives itératives ont pu avoir lieu, cependant elles sont en quelque sorte restées « privées » à chaque responsable, qui rend compte de ses conclusions dans des documents officiels. Dans la méthode agile*, cela n'est même pas vraiment le cas, puisqu'elle encourage le dialogue permanent entre collaborateurs, de telle sorte qu'ils puissent construire la solution finale ensemble. Ce qui diffère donc vraiment entre la programmation professionnelle et amateur, ces sont les « points de rendez-vous » explicites que se donne la première pour trouver des points d'accord progressifs documentés (d'où l'impression de linéarité). Tous ces points d'étapes sont révisables. Par exemple le codage d'une fonction peut se révéler trop ardu, ou trop peu efficace : le développeur peut alors revenir vers le responsable de la spécification pour lui demander de revoir son schéma. Cependant, il y a ici l'idée implicite que cela ne devrait survenir que dans des cas exceptionnels, et que le projet va avancer par des *engagements progressifs*, dont les premiers sont de plus en plus difficiles à remettre en cause au fur et à mesure du progrès. Il s'agit donc d'un travail de *convergence planifiée*, qui n'exclut cependant jamais sa propre reconstruction plus ou moins radicale au cours du processus.

Ensuite, on perçoit à travers le style d'écriture la présence de conventions et de standards. Il y a très peu de commentaires, et l'intention du programmeur est principalement véhiculée par les conventions de nommage. Ainsi, même si une

valeur n'est utilisée qu'une seule fois, elle est tout de même affectée à une variable, par exemple `currentIndex` ligne 9. On dit que le style est verbeux* (*verbose*), ce qui n'est pas péjoratif. Cela pourrait être traduit par *explicite* en langage courant. L'objectif est d'améliorer la lisibilité du code en explicitant ce à quoi sert cette valeur. Par ailleurs, cette fonction réalise une tâche unique et dépend donc d'un contexte local (la requête en cours) et global (la base de données des salles) qui sont supposés connus et qui structurent fortement le code. Ces pratiques (nommage explicite des variables, unicité de visée de la fonction) sont conformes aux normes de qualité actuelles du développement logiciel, telles qu'on les trouve par exemple chez R. C. Martin (2009) et D. Thomas et Hunt ([1999] 2020).

Enfin, cette fonction est très simple au niveau algorithmique. Le travail de la réflexion n'a pas consisté ici, comme dans l'exemple de § 28, à chercher une formule de récurrence ou à optimiser un temps de traitement. La complexité se situe ailleurs, dans l'architecture globale de l'application qui lui permet de traiter des requêtes comme celle-ci avec des fonctions aussi simples.

§ 34. La vision artisanale

Ce panorama rapide de la programmation professionnelle peut donner l'impression d'une convergence progressive vers un ensemble de pratiques professionnelles matures, codifiées par les associations industrielles et structurant les curricula de formation.

Cependant, ce n'est pas là la vision qu'en donnent Jacobson et al. (2019), qui militent depuis une dizaine d'années en faveur de l'adoption d'une telle codification, qu'ils appellent **Essence**, et sur laquelle nous allons revenir. Ils dressent un tableau plutôt noir de l'état de l'ingénierie logicielle, qui serait victime de guerres de chapelles entre gourous des méthodes :

> En général, chaque méthode reconnue a un père fondateur, et parfois plus d'un. En cas de succès, ce père fondateur est élevé au rang de gourou. Le gourou dicte plus ou moins ce qui entre dans sa méthode. Ainsi, une fois que vous avez adopté une méthode, vous avez l'impression d'être dans une prison contrôlée par le gourou. [Il s'agit] d'une situation indigne de n'importe quelle industrie et en particulier d'une industrie aussi énorme que celle du logiciel. […]
>
> L'ingénierie logicielle est aujourd'hui gravement entravée par des pratiques immatures. Les problèmes spécifiques incluent :
>
> 1. La prévalence de modes plus typiques de l'industrie de l'habillement que d'une

discipline d'ingénierie.
2. L'absence d'une base théorique solide et largement acceptée.
3. Le nombre considérable de méthodes et de leurs variantes, avec des différences peu comprises et artificiellement amplifiées.
4. L'absence d'évaluation et de validation expérimentales crédibles.
5. Le clivage entre la pratique industrielle et la recherche universitaire[56].

Ce tableau est si noir qu'on peut se demander s'il est entièrement sincère. Jacobson avoue avoir été lui-même l'un de ces gourous au début des années 2000, puisqu'il est l'un des pères fondateurs d'UML. Un lecteur malveillant pourrait ainsi interpréter son pessimisme comme motivé par le dépit éprouvé à voir ce standard supplanté par les méthodes agiles* et les nouvelles techniques de programmation, et par sa tentative d'imposer un nouveau standard, Essence, dans le monde de l'ingénierie logicielle.

Cependant, ce diagnostic touche juste, et le foisonnement des méthodes agiles* en est un symptôme. La difficulté d'adapter les principes très purs du *Manifeste agile*, comme ceux évoqués ci-dessous, à la réalité des grands projets industriels, a donné lieu à quantités de cadres méthodologiques, comme Crystal, Extreme Programming, Scrum. Plus généralement, la fragmentation des méthodes est liée à l'*idiosyncrasie* de tout project logiciel. Comme le dit Jacobson :

> Il y a [aujourd'hui] environ 20 millions de développeurs de logiciels dans le monde et leur nombre augmente d'année en année. On peut estimer qu'il existe plus de 100 000 méthodes différentes de développement, puisque chaque équipe a développé sa propre méthode de travail, même si elle ne l'a pas décrite explicitement [57].

Cette hypothèse est convaincante. Étant donné tous les facteurs cités plus haut qui peuvent se combiner pour structurer l'environnement de travail d'un projet – la pile technologique*, le poids de la dette informatique*, les outils à disposition, la culture du projet, son contexte économique (par exemple logiciel libre *vs* commercial), les personnalités des directeurs de projet et des architectes, etc., les manières de travailler sont sans doute très individualisées par équipe. Une étude récente avance par exemple qu'il ne faut par surestimer l'importance des méthodes explicites dans la conduite pratique des projets :

> Notre analyse montre que, dans les deux cas, la variation induite par les méthodes agiles* et modèle en cascades* représente environ 40% de toutes les activités, tan-

56. (Jacobson et al. 2019, p. 27-28).
57. (ibid., p. 27).

> dis que les 60% restantes peuvent être expliqués par les habitudes personnelles du concepteur, les conditions d'adéquation du projet à la méthode et le bruit ambiant. En général, l'effet de la méthode sur les activités de conception de logiciels est plus faible que ce que l'on croit. L'influence du concepteur et des conditions du projet sur les processus et les résultats ne doit donc pas être sous-estimée[58].

En particulier, la dimension humaine et la motivation personnelle des développeurs, qui s'expriment souvent par la culture du hacking* et ses dimensions ludique, virtuose et compétitive, entrent en conflit avec l'approche normative et industrielle de l'ingénierie logicielle.

D'une certaine manière, le mouvement agile* naît d'un instinct *anti-méthodique*, qui réagit au poids des normes édictées par la programmation structurée* et la programmation objet*. Certains des « commandements » du *Manifeste agile* laissent clairement transparaître cette sensibilité « libertaire » :

> — Réalisez les projets avec des personnes motivées. Fournissez-leur l'environnement et le soutien dont ils ont besoin et faites-leur confiance pour atteindre les objectifs fixés.
> — La méthode la plus simple et la plus efficace pour transmettre de l'information à l'équipe de développement et à l'intérieur de celle-ci est le dialogue en face à face.
> — Un logiciel opérationnel est la principale mesure d'avancement.
> — Les meilleures architectures, spécifications et conceptions émergent d'équipes auto-organisées[59].

La promotion d'idées d'auto-organisation, de dialogue face-à-face, de confiance et de motivation, et corrélativement la dévaluation de celles de documentation et de mesure de performance, sont clairement dirigées contre le principe même d'une approche industrielle de la programmation. La notion d'*artisanat* (*craftsmanship*) est associée au mouvement agile* dès sa conception, notamment dans l'ouvrage de McBreen (2001). Elle s'exprime par la valorisation des dimensions d'autonomie de l'individu, de confiance et de compagnonnage dans le travail en équipes, de plaisir et de travail bien fait. Plutôt que des méthodes, elle se contente d'énoncer des conseils ou des « tuyaux », qu'elle illustre par des exemples concrets et des anecdotes, comme les livres de R. C. Martin (2009) et D. Thomas et Hunt ([1999] 2020) par exemple. McBreen va plus loin et affirme que l'écriture de bons logi-

58. (Thummadi et Lyytinen 2020, p. 864).
59. (Beck et al. 2001).

ciels est incompatible avec une approche mécanique et standardisée du processus de développement. Il écrit par exemple :

> Une faille essentielle de la métaphore de l'ingénierie logicielle est qu'elle échoue à placer les gens au centre du processus de développement. […]
> Plutôt que d'essayer d'industrialiser et de déqualifier le développement, l'approche artisanale du logiciel (*software craftsmanship*) consiste à acquérir la maîtrise. Oui, une personne peut apprendre à exécuter un sous-ensemble de tâches en un temps relativement court, mais le développement d'une maîtrise complète ne se fait pas rapidement[60].

Ce parti-pris artisanal au sein de la profession est ancienne. Déjà Peter Naur, dans les années 1980, s'était élevé contre la direction industrielle que prenait la programmation structurée*. Dans un texte sur lequel nous revenons plus loin, il présente deux « philosophies » de la programmation : la première, à laquelle il adhère, est fondée sur la compréhension par le programmeur du problème général que son application doit aider à résoudre et du contexte dans lequel elle va être utilisée. Sa compétence essentielle est de pouvoir relier le code de son programme au « monde » du problème :

> Le programmeur qui possède la théorie du programme peut expliquer comment la solution est liée aux affaires du monde qu'elle aide à traiter. Une telle explication devra porter sur la manière dont les affaires du monde, tant dans leurs caractéristiques générales que dans leurs détails, sont, d'une certaine manière, transposées dans le texte du programme et dans toute documentation supplémentaire. […] La plus grande partie des aspects et des activités du monde se situera bien sûr en dehors du champ d'application du texte du programme, n'étant pas pertinente dans le contexte. Cependant, la décision qu'une partie du monde est pertinente ne peut être prise que par quelqu'un qui comprend le monde entier. Cette compréhension doit être apportée par le programmeur[61].

Dans cette conception, le programmeur n'est pas seulement un spécialiste informatique, il doit savoir devenir également spécialiste de la discipline qu'il assiste et acquérir au moins une partie de son savoir. Une telle approche favorise l'accumulation de l'expérience d'un programmeur autour d'un même logiciel, ou au moins autour d'un domaine spécifique d'activité. Selon Naur, c'est cette continuité dans le temps des personnes et des équipes qui est fondamentale pour résoudre le problème de la maintenance informatique (nous y revenons plus loin, § 91). Elle exige

60. (McBreen 2001, p. 31, 34).
61. (Naur 1985, p. 256).

donc une attention aux personnes et à la gestion de leur carrière plutôt qu'aux méthodes et aux outils.

Il oppose cette conception à la vision industrielle pour laquelle l'activité d'un programmeur est consiste seulement à écrire du bon code :

> Une grande partie de la discussion actuelle sur la programmation semble supposer qu'elle est similaire à la production industrielle, le programmeur étant considéré comme un composant de cette production, qui doit être contrôlé par des règles de procédure et qui peut être facilement remplacé [...] Au niveau de la gestion industrielle, ces points de vue soutiennent le fait de traiter les programmeurs comme des travailleurs ayant un niveau de responsabilité assez faible et n'ayant reçu qu'une formation brève[62].

Pour Naur comme pour McBreen, un bon développement logiciel ne dépend pas d'une bonne méthode, mais d'un bon développeur, car la programmation est pour eux une activité épistémique de résolution de problème qui exige d'analyser et de *comprendre* la spécificité des problèmes en jeu. Cette compréhension, acquise par l'expérience de problèmes similaires et le compagnonnage, et la seule qui peut espérer atteindre la *vraie* systématicité, celle qui est spécifique au problème à résoudre, et qui ne peut être résumée par une quelconque formule méthodique.

§ 35. Le projet Essence

Jacobson prend le contrepied de ces idées, tout en cherchant à leur faire une place dans sa vision. Si les méthodes agiles* ont selon lui le mérite d'avoir mis en valeur les dimensions humaines de la programmation, il est hors de question de leur sacrifier la rigueur et la sécurité qu'apportent les méthodes industrielles :

> Passer du développement à l'ingénierie requiert de s'appuyer davantage sur la science et moins sur l'artisanat, ce qui se traduit généralement par une certaine manière de décrire une méthode de travail spécifique et par une plus grande automatisation. Cela permet la répétabilité et la cohérence d'un projet à l'autre. L'ingénierie signifie également que les équipes, par exemple, apprennent en travaillant et améliorent continuellement leur façon de travailler.[...]
>
> Il est vrai que certains produits merveilleux [...] ont autrefois été développés par un ou quelques individus dotés d'une grande vision et de la programmation comme seule compétence. Cependant, à présent que la grande vision est mise en œuvre, soyez certains que ces entreprises ne comptent plus sur des programmeurs

62. (Naur 1985, p. 260).

> héroïques. Aujourd'hui, ces entreprises ont embauché les meilleurs talents, qui ont une longue expérience en ingénierie logicielle[63].

Jacobson ne cherche pas à promouvoir un nouveau standard d'excellence concernant l'ingénierie logicielle, mais à proposer une « méta-méthode », un cadre permettant de décrire de manière homogène toute méthode de développement, de manière à les rendre interopérables entre elles. Pour lui, en effet, en-deçà des modes méthodologiques se trouve un fond commun de principes et de mini-pratiques dont un sous-ensemble seulement est pertinent pour un contexte donné, et qui forme la réalité de la « méthode » de l'équipe. Le cadre Essence vise à permette la description uniforme de ces méthodes particulières de manière à pouvoir les rendre interopérables.

Essence propose de décrire chaque méthode comme une combinaison de pratiques, elles-mêmes définies à partir 1) d'activités, comme une réunion quotidienne, 2) de résultats, comme un document de spécifications techniques, et 3) de compétences, comme la connaissance d'une technologie. Mais les éléments les plus importants sont les *alphas* qui permettent d'évaluer le progrès et la « santé » du projet.

Alpha est un terme abstrait choisi à dessein pour éviter sa confusion avec les activités ou les résultats du projet. Il y a sept *alphas* présents en tout projet d'ingénierie logicielle : 1) le *système logiciel* réalisé, qui doit satisfaire 2) une *expression de besoins*, eux-même justifiés par des 3) *opportunités* ou enjeux qui intéressent 4) des *parties prenantes* (clients, utilisateurs, etc.). 5) Le *travail de développement* est réalisé par 6) une *équipe* ayant 7) une *manière de travailler*.

Une telle présentation semble enfoncer des portes ouvertes et ne pas dire grand' chose par elle-même. Cependant, l'intérêt de cette liste est de mettre en évidence qu'on a affaire ici à des objets *intentionnels* qui projettent des attentes sur le projet. Les *alphas* sont autant de perspectives grâce auxquelles on peut s'interroger sur la santé du projet et de son progrès. Une équipe par exemple n'est pas seulement une somme d'individus. Pour savoir si elle est en mesure de réussir le projet, il faut se demander si elle a la taille suffisante, les bonnes compétences, la capacité à bien travailler ensemble et à s'adapter à des situations changeantes[64].

Pour prendre un second exemple, le *système logiciel* n'est pas seulement un processus informatique, une application sur un mobile ou un listing de code. Il

63. (Jacobson et al. 2019, p. 33-35).
64. (ibid., p. 48).

peut être un ensemble complexe de choses de ce type, mais également des interfaces, des droits d'accès, des bases de données, etc. Toutes ces choses sont dans le système mais il ne se réduit pas à elles. Il se définit d'abord relativement aux attentes qu'il doit satisfaire. Il doit également avoir une certaine robustesse technique (taux d'erreur, vitesse, capacité à gérer les pics de charge, avoir une interface aisée à comprendre, etc.) – des attentes qui sont souvent implicites à l'expression de besoin. Il soit surtout être évolutif et adaptable[65] – un point que nous avons déjà développé[66], et qui permet de voir pourquoi, en droit, le système logiciel ne peut être entièrement asservi à l'expression de besoins – pourquoi il doit répondre à des normes techniques indépendantes.

Une méthode peut définir un *alpha* qui lui est spécifique. Par exemple dans la méthode agile* nommée Scrum, Jacobson argumente que le sprint doit être compris comme un *alpha*. Le sprint est en effet une phase de développement courte (habituellement quinze jours) visant la livraison par une petite équipe d'une liste de fonctionnalités déterminée. Il s'agit d'une abstraction visant à organiser et à mesurer le progrès d'une petite équipe, c'est donc bien un *alpha*, qui se rattache à l'*alpha* élémentaire du travail de développement.

Jacobson montre comment, de manière similaire, on peut définir dans le cadre d'Essence d'autres méthodes, comme celle de la user story, une manière populaire aujourd'hui d'expliciter les spécifications, ou encore celle des micro-services*, une architecture logicielle[67]. Ainsi, un même projet peut faire appel à différentes méthodologies en fonction de son contexte d'usage, des technologies utilisées, de l'organisation des équipes. Le besoin de mise en cohérence des activités et des livrables intermédiaires dans une planification opérationnelle explique, selon Jacobson, que de nouvelles méthodes particulières voient sans cesse le jour, mais qui ne sont en fait que des re-combinaisons de méthodes existantes. En proposant un cadre uniforme pour les décrire toutes, Jacobson avance qu'il met à disposition de chaque équipe l'outillage conceptuel nécessaire pour effectuer elle-même cette combinaison en l'adaptant exactement à ses besoins, lui évitant ainsi de se soumettre à celle d'un gourou particulier, qui ne taillera jamais exactement le vêtement adéquat. Cela lui permet, en reprenant de manière paradoxale le leitmotiv du mouvement agile*, d'affirmer que sa méta-méthode permet de « libérer les pratiques des pri-

65. (Jacobson et al. 2019, p. 46).
66. Voir plus haut, § 30.
67. (Jacobson et al. 2019, partie 2).

sons des méthodes[68] », tout en faisant l'apologie d'une approche industrielle du développement logiciel !

Il ne nous appartient pas de juger la pertinence pratique de ces propositions. Il semble qu'elles rencontrent un succès au moins d'estime, y compris de la part des pionniers de l'agile*, cependant leur déploiement effectif au service de la gestion de projet est encore limité.

L'intérêt de ce cadre d'analyse pour notre propos réside plutôt dans son intention même de trouver une solution pérenne à une tension qui semble intrinsèque à la programmation, entre le besoin de systématicité et de rigueur d'une part (l'approche « industrielle ») et l'incroyable diversité et évolutivité des conditions de son exercice. Il nous semble – même s'il faudrait faire des enquêtes complémentaires pour étayer cette affirmation – que nulle autre discipline d'ingénierie n'a *vécu* cette tension comme un tel drame. Après tout, toutes exhibent une pluralité de méthodes (souvent différentes entre pays) et des manières de faire idiosyncrasiques sans que ces différences soient perçues comme vitales pour la réussite des projets. Par opposition, le débat concernant les méthodes de programmation a toujours pris un ton passionné et polémique, et cela depuis les imprécations de Dijkstra contre l'instruction `Goto` dans les années 1960[69], comme si la méthode de travail était l'âme même du produit. Ce fait semble pointer vers cette réflexivité essentielle qui caractérise la programmation, celle du besoin d'*être systématique* soi-même afin de trouver *une solution systématique* à un problème, – quoique ces expressions veuillent dire pour l'instant. On peut d'ailleurs noter que le cadre d'**Essence** est si général qu'il pourrait s'appliquer à tout projet de développement d'un produit quelconque, et pas seulement à du logiciel.

À ce titre, une proposition curieuse de Jacobson nous semble révélatrice : au chapitre 7 de l'ouvrage étudié, il avance qu'**Essence** est une *théorie* du développement logiciel. Or les sciences de l'ingénieur (*engineering sciences*), malgré la difficulté à déterminer leurs contours exacts[70], et même si elles sont des sciences du *comment faire ?*, ne se donnent cependant jamais pour objet principal leurs propres méthodes de travail. Par exemple, l'ingénierie chimique étudie comment réaliser tel composé chimique, comment réaliser telle propriété matérielle en évitant telle autre, etc. mais elle n'étudie pas la manière dont on a trouvé ces méthodes, si-

68. (ibid., p. xxv).
69. (Dijkstra 1968).
70. Voir (Michelfelder et Doorn 2020, ch. 6 et 7) sur ces sujets.

non de façon adventice, dans des considérations épistémologiques annexes. Or c'est bien cela que propose Jacobson, de définir la science de l'ingénierie logicielle comme la science de ses méthodes de travail, et non de ses objets (qui seraient par exemple les algorithmes, les bases de données, les protocoles de communication). Un tel « décalage réflexif », qui semble situer la programmation dans le domaine du « méta- », n'est pas accidentel ; il l'apparente plutôt à ce qu'on appelle aujourd'hui la science du *design*, un sujet sur lequel nous revenons au chapitre 4.

Au terme de cet examen de la programmation professionnelle, il nous semble impossible de prendre un chemin axiologique pour définir la programmation : malgré ses acquis pratiques indéniables, l'ingénierie logicielle ne semble pas pouvoir être réduite à une méthode particulière, car elle dépend, au moins dans le cas général, d'une forme d'intelligence situationnelle du programmeur qui consiste à justement réduire la diversité et la confusion du problème à résoudre à la clarté d'une solution systématique – une tâche qui ne peut elle-même être guidée systématiquement, mais au mieux être encadrée par des garde-fous lui évitant des erreurs communes. Et lorsqu'on cherche, comme le fait Jacobson avec Essence, à abstraire un langage universel commun à toutes les méthodes de programmation, il semble qu'on se trouve réduit à des notions si générales qui, quoiqu'elles puissent être utiles en pratique, ne peuvent en aucun cas fonder une *théorie* de la programmation.

2.4 La programmation dans tous ses états

Dans les trois sections qui suivent nous développons une approche pour définir la programmation par ses caractéristiques essentielles, dont le noyau pourrait être cette expression de Welsh, qu'elle consiste à « diriger les machines à faire ce qu'on veut ».

Nous commençons donc par revenir, dans cette section, à la diversité des formes de la programmation, afin de prendre la mesure la plus large du spectre que notre définition devra recouvrir. Le point essentiel de cette diversité, on se le rappelle, est qu'elle est dynamique. Aussi, plutôt que tenter une taxonomie des formes de programmation, nous essayons de dégager les quelques *directions* dans lesquelles la programmation semble se transformer (§ 36).

Nous illustrons plus en détail deux d'entre elles, d'abord celle où la programmation tend à s'assimiler à d'autres activités professionnelles en adoptant leur lo-

gique propre (§ 37), ensuite celle où elle tend à se simplifier jusqu'à se fondre dans l'usage de logiciels plus intuitifs, notamment visuels (§ 38). La section suivante 2.5 est consacrée à la proposition d'une définition générique de la programmation, et la dernière (section 2.6), à son examen critique.

§ 36. La programmation alternative

Il ne s'agit pas ici de proposer une taxonomie des programmes ou des formes de programmation, une tâche entreprise de nombreuses fois avec des résultats aussi intéressants que chaque fois différents. On peut en effet classer les programmes selon leur fonction, leur taille, leur niveau d'interactivité, leurs paradigmes de programmation*, etc. Le groupe de recherche PROGRAMme[71] doit publier une revue de ces taxonomies et nous n'y revenons pas.

Nous cherchons ici plutôt à capturer les manières selon lesquelles la programmation s'infiltre et se dissout dans d'autres activités – ainsi que nous l'avons affirmé avec Haigh[72], et par-là même devient méconnaissable. En d'autres termes, nous nous intéressons aux axes de transformation de la programmation qui l'amènent à prendre les formes les plus éloignées de notre entente courante de cette activité, et qui vont donc être les plus difficiles à ramener à l'unité d'une définition.

Cette entente courante est, pour le dire rapidement, celle d'une activité dont le résultat est un texte écrit dans un langage de programmation de haut niveau dit impératif*, comme Python, Java ou C, où le processus informatique peut être « lu » dans la séquence d'instructions déroulées dans le texte, *modulo* des instructions de branchement conditionnel, des sauts, ou des appels de procédures. Dans cette entente courante nous incluons également les autres paradigmes de programmation* courants, comme la programmation fonctionnelle*, la programmation objet* ou la programmation concurrente*.

Nous envisageons ici quatre directions différentes par lesquelles la programmation peut s'éloigner de ce modèle dominant :

1. La programmation peut devenir si spécialisée à un domaine très précis de problèmes, et en emprunter si bien le langage et la logique, qu'elle ne semble plus du tout avoir à faire avec des ordinateurs ni même des machines.
2. La programmation, en cherchant à se démocratiser, peut se simplifier jusqu'à ressembler à un usage avancé de l'ordinateur, voire à un jeu.

71. *What is a computer program ?*, à paraître.
72. Voir plus haut, § 25.

3. L'apprentissage machine* (qu'on pourrait aussi appeler *programmation par apprentissage*) « apprend » à l'ordinateur à résoudre des problèmes à l'aide d'exemples de résolution. Le programmeur est donc ici ce qu'on appelle actuellement un *ingénieur de données* (*data scientist* ou *data engineer*) voire, depuis l'apparition de ChatGPT, un *invite de commande** *engineer* dont l'expertise consiste à bien cadrer, en langage naturel, les demandes adressées à l'intelligence artificielle[73]. C'est cette forme de programmation que Welsh[74] voit succéder à la programmation telle que nous la connaissons.
4. On peut programmer des machines qui ne se laissent pas décrire selon le modèle classique de l'ordinateur, dit *de Von Neumann*. L'exemple le plus connu est la programmation d'ordinateurs analogiques* qui pré-date l'apparition de l'informatique classique[75]. Cependant de nouvelles formes de « machines programmables » sont régulièrement conçues, dont le matériau élémentaire n'est plus le circuit électrique. On peut d'une manière générale avancer que la recherche explore la programmabilité *de toute forme de matière régie par des lois spécifiques*. On parle ainsi d'ordinateur quantique, de programmation moléculaire, de programmation de matériaux déformables, ou encore de programmation bionique qui peut porter sur des populations bactériennes ou sur des macro-organismes comme des insectes dont on contrôle le système nerveux.

Nous choisissons d'illustrer les deux premières directions de métamorphose de la programmation. Nous ne développons pas ici l'analyse des deux suivantes pour deux raisons. D'abord, elles sont plus techniques et nous entraîneraient dans de trop longs développements. Surtout, nous pensons qu'elles ont en effet des implications majeures sur toutes nos questions qu'elles envisagent à leurs limites. L'apprentissage machine* remet en cause ce que nous entendons par *résolution de problème* et les nouveaux modèles d'ordinateurs remettent en cause ce que nous entendons par *machine*. Aussi pensons-nous qu'il n'est possible, à ce stade préliminaire de notre réflexion, que de réduire d'une *manière nominale* ces formes nouvelles de programmation à l'unité d'une définition. Nous ne pourrons y revenir qu'à la fin de notre parcours, lorsque nous aurons établi de manière plus ferme ce que programmer veut dire.

73. (Quilty-Harper 2023).
74. Voir plus haut, § 25.
75. (Bournez et Pouly 2021, p. 177).

Il est par contre plus aisé et intéressant à ce stade de donner des exemples de spécialisation (§ 37) et de simplification (§ 38) de la programmation, qui font écho aux mouvements d'ouverture et de fragmentation de la profession que nous décrivions tout à l'heure[76]. Les formes simples de programmation, en effet, visent à l'ouvrir à des populations toujours plus larges, et les formes spécialisées à s'adapter aux besoins spécifiques de professionnels dont le métier principal n'est pas le développement logiciel.

§ 37. Formes spécialisées de la programmation

La programmation spécialisée s'appuie souvent sur des langages dits *de spécification*, c'est-à-dire qui semblent se désintéresser de la « direction de la machine » pour se concentrer sur la description exacte du problème à résoudre. Un bon exemple ici est SQL, le « langage » le plus répandu de traitement des bases de données, dont on se demande souvent s'il permet de « vraiment » programmer[77]. Un exemple de code SQL sur une base de données prédéfinie, ainsi que son résultat, est donné en 2.7.

Code 2.7 – Exemple de code SQL et son résultat.
Source : (Bowman, Emerson et Darnovsky 1996, p. 179)

```
1  select type, avg(price)
2  from titles
3  where advance > 5000
4  group by type
5  order by 2
6
7  Résultats:
8  type
9  ----------------------------
10 business          2.99
11 mod_cook          2.99
12 psychology       14.30
13 trad_book        17.97
14 popular_comp     21.48
```

Cette requête porte sur une base de données de livres, appelée `titles`, dont chaque élément est entre autres caractérisé par un prix (`price`) et une catégo-

76. Voir plus haut, § 27.
77. Voir par exemple (Koydan 2020).

rie (`type`), par exemple cuisine, affaires, psychologie. Le programme est écrit sur les lignes 1 à 5 et le résultat (illustratif) de son exécution est affiché lignes 10 à 14. La requête, introduite par le mot-clé `select` (ligne 1), demande à sélectionner, parmi les titres dans la base (ligne 2), ceux qui ont bénéficié d'une avance de plus de $5000 (`where`, ligne 3), à les regrouper par catégorie (`group by`, ligne 4) en affichant celle-ci ainsi que leur prix moyen (ligne 1) par ordre croissant du prix (ligne 5, `2` désigne la deuxième colonne).

Le caractère étrange de ce « programme » est qu'il n'est doté que d'une seule « instruction », `select`, et que l'essentiel de la programmation consiste à spécifier le contenu de la sélection souhaitée ainsi que ses modalités de mise en forme. Ces modalités peuvent inclure des calculs, comme ici la moyenne. On ne s'intéresse par contre pas du tout à la manière dont l'ordinateur va constituer cette sélection en parcourant la base de données – ce qui habituellement est le travail du programmeur. SQL est une forme de programmation logique*, au même titre que Prolog, où on ne s'intéresse qu'à la spécification des contraintes que doit satisfaire le résultat du processus informatique.

Le point essentiel, nous semble-t-il, est d'inviter le programmeur à prendre résolument le point de vue de la *spécification*, c'est-à-dire du problème à résoudre, et non celui du fonctionnement de la machine, qui est autant que possible masqué et mis de côté.

On trouve dans cette forme de programmation de nombreux langages visant à s'approcher le plus possible des logiques internes de disciplines particulières. Nous avons déjà mentionné les langages de conception de circuits intégrés, comme VHDL. Il y a également les langages permettant de modéliser les télécommunications (Erlang), les raisonnements mathématiques (Coq, Julia), les structures architecturales (Grasshopper, Dynamo), la musique (CSound), Tous ces langages sont destinés à des praticiens de ces disciplines, ou à des programmeurs devant interagir intensivement avec ces derniers. Deux exemples détaillés vont nous permettre d'illustrer cette possibilité de la programmation.

Nous prenons d'abord l'exemple d'un langage destiné aux ingénieurs, Modelica. En ingénierie mécanique et industrielle, la simulation dynamique de systèmes est devenue si prégnante qu'elle y devient programmation à part entière. La figure 2.3 est l'interface visuelle d'un programme écrit dans le langage Modelica, qui permet de simuler le comportement d'un système d'air conditionné[78]. Modelica est

78. (Wetter et al. 2011).

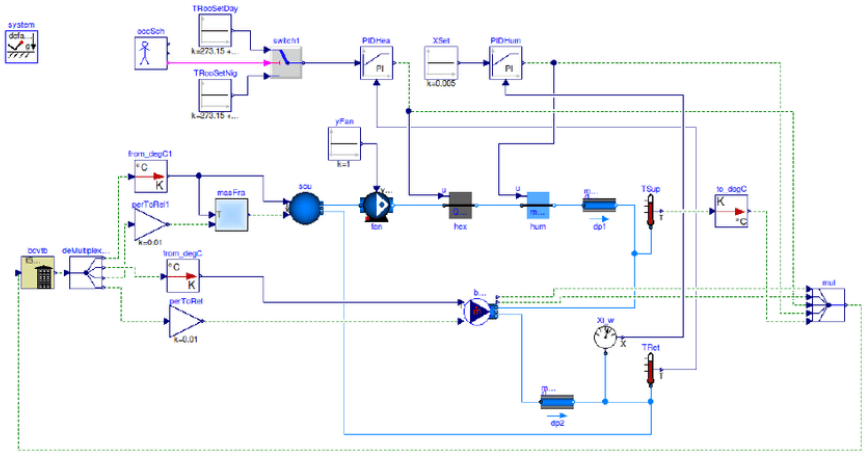


FIGURE 2.3 – L'interface visuelle de Modelica
Source : (WETTER et al. 2011)

un langage conçu pour les ingénieurs, et est une bonne illustration de la convergence entre programmation et conception industrielle qu'avait prédite Simon[79]. Programmer, ici, revient à expliciter le système des règles que doivent satisfaire les phénomènes étudiés. Chaque élément est un objet ayant des propriétés physiques (par exemple énergie, chaleur, eau ou électricité) qui sont régies par des équations. Les connecteurs entre objets permettent de décrire les relations de contraintes entre composants. Les entrées du programme sont les conditions initiales du système. L'exécution du programme déclenche un algorithme de résolution de contraintes qui détermine en sortie les comportements temporels de tous ses observables, ceux-ci pouvant faire l'objet d'une simulation graphique.

Modelica dispose de librairies de composants physiques (circuits électriques, panneaux solaires, pièces mécaniques, pompes à chaleur, etc.) qui peuvent être assemblés librement pour construire des systèmes dynamiques et des vues globales de leurs comportements, de sorte qu'un objet peut avoir un comportement thermique, un comportement électrique, etc. qui peuvent également interagir les uns avec les autres. La programmation peut être abordée par la réutilisation et la disposition visuelle de ces composants dans l'interface déjà illustrée ci-dessus.

Les contraintes des composants de base sont indiquées dans un code d'apparence classique, mais qui fondamentalement exprime des équations comme le ferait un physicien. En 2.8, on voit le code Modelica d'une balle rebondissante, soumise

79. Voir plus haut, section 4.2.

seulement à la gravité g, lâchée à la hauteur h et ayant un coefficient d'élasticité au rebond e.

CODE 2.8 – Code Modelica pour une balle rebondissante

```
1   model BouncingBall "The 'classic' bouncing ball model"
2   type Height=Real(unit='m');
3   type Velocity=Real(unit='m/s');
4   parameter Real e=0.8 "Coefficient of restitution";
5   parameter Real g=-9.81 "gravity";
6   parameter Height h0=1.0 "Initial height";
7   Height h "Height";
8   Velocity v(start=0.0, fixed=true) "Velocity";
9   initial equation
10  h = h0;
11  equation
12  v = der(h); "derivative"
13  der(v) = -g;
14  when h<0 then
15  reinit(v, -e*pre(v));
16  end when;
17  end BouncingBall;
```

Les lignes 2 à 8 de ce code définissent les grandeurs pertinentes, les constantes (`parameter`) et les variables du système, les lignes 9 et 10 son équation initiale, et les lignes 11 à 16 son équation invariante, qui agissent comme des contraintes sur la recherche de valeurs convenant aux deux variables définies (hauteur et vitesse) en fonction du temps.

On voit ici que la « programmation » se fait dans le langage naturel du physicien. Il n'y a pas vraiment d'instructions dans Modelica, ni même de définition de fonctions, si bien que plusieurs informaticiens, dans un échange public, ont nié que Modelica soit un « vrai » langage de programmation[80]. Selon eux, il s'agissait d'un logiciel de simulation.

Un second exemple est le langage Catala qui vise à programmer les règles du droit social et fiscal servant au calcul de l'impôt ou des droits sociaux, et ayant donc une forte composante algorithmique. Catala vise à rendre le plus aisé possible le dialogue entre le programmeur et le juriste qui doit vérifier le codage des règles. La syntaxe du langage tente ainsi d'épouser au plus près la logique du droit, qui

80. Échange au cours du groupe de travail PROGRAMme.

raisonne par cas généraux et par exceptions. En voici un court exemple, extrait du code de la sécurité sociale :

CODE 2.9 – Programmation Catala de l'article L521-3 du code civil, (MÉRIGOUX 2021)

```
1  "Chacun des enfants à charge, à l'exception du plus âgé, ouvre droit
       à partir d'un âge minimum à une majoration des allocations
       familiales."
2  champ d'application CalculAllocationsFamiliales :
3    règle majorations_allocations_familiales.droits_ouverts de enfant
4    sous condition
5      (enfant dans ménage.enfants) et
6      (non (est_enfant_le_plus_âgé de enfant)) et
7      (enfant.âge ≥ 1521_3.âge_limite alinéa_1 de enfant)
8    conséquence rempli
9
10 "Toutefois, les personnes ayant un nombre déterminé d'enfants à
       charge bénéficient de ladite majoration pour chaque enfant à
       charge à partir de l'âge mentionné au premier alinéa."
11
12 champ d'application CalculAllocationsFamiliales :
13   règle majorations_allocations_familiales.droits ouverts de enfant
14   sous condition
15     (enfant dans ménage.enfants) et
16     (nombre de ménage.enfants ≥ 1521_3.minimum_alinéa_2) et
17     (enfant.âge ≥ 1521_3.âge_limite_alinéa_1 de enfant)
18   conséquence rempli
```

Dans cet exemple, le code (lignes 2-8, 12-18) alterne avec le texte de loi (lignes 1, 10) qu'il modélise, afin de faciliter la comparaison directe des deux formulations. La syntaxe de Catala imite celle des textes de lois, qui procède par l'énoncé d'une règle générale puis gère des exceptions, et elle nous semble en effet assez claire pour ne pas requérir d'explication complémentaire. Un tel texte est cependant un programme, puisqu'il se compile en C, JavaScript ou R et par là permet de calculer effectivement les allocations familiales d'un ménage. Dans un commentaire intéressant fait à l'oral, Mérigoux, le concepteur de ce langage, remarque que la langue des juristes est étonnamment concise et tient à peu près autant de place que le code informatique correspondant : on peut le vérifier ici, si on compte le nombre de mots utilisé par chaque formulation. Cela suggère que cette langue aurait elle-même adopté une forme procédurale.

§ 38. Programmation simplifiée et visuelle

La seconde forme de métamorphose de la programmation que nous voulons illustrer est celle de sa simplification, qui rend difficile sa distinction avec l'usage d'applications complexes. Nous commençons par l'exemple du tableur* qui a mis à disposition du plus grand nombre la programmation de calculs chaînés, dont la complexité peut être grande. Un tableur est constitué de cellules dans lesquelles des valeurs (nombres, texte) peuvent être entrées, ou des formules décrivant des fonctions dépendant du contenu d'autres cellules.

On peut par exemple considérer la formule suivante (2.10). Selon que l'amplitude du taux de croissance entre la valeur de la cellule nommée `D8` et celle de `C8` est inférieur à 20% ou non, elle enregistre dans la cellule qui lui est associée la valeur de la cellule E8 ou celle-ci divisée par 2. Cela peut être utile, par exemple, afin d'appliquer un taux de subvention différents à deux entreprises, selon qu'elles croissent très rapidement ou non. S'il y a une erreur dans le calcul de cette formule (notamment si `C8` a pour valeur 0), le texte `NA` est renvoyé.

CODE 2.10 – Formule Excel illustrative

```
=SIERREUR(SI(ABS(D8/C8-1)<20%;E8;E8*0,5);"NA")
```

Un tableur est capable aussi de calculer des formules itératives, du moment qu'on consent à les représenter spatialement, par exemple en colonnes ou en lignes. La récursivité peut être approchée par des itérations. Par exemple, la feuille de calcul 2.4 calcule de manière itérative la fonction factorielle. Les noms des cellules sont donnés par les formulations `LyCx` où `y` est le numéro de la ligne et `x` celui de la colonne. `L2C1` et `L2C2` sont initialisées à 1. Les cellules des lignes suivantes `3` à `12` contiennent une même formule sur chaque colonne `C1` et `C2`, indiquée respectivement à la première et à la seconde ligne du code § 38. La première formule, pour les cellules `L3C1` à `L13C1`, ajoute `+1` à la cellule située sur la ligne précédente, indiquée par `L(-1)C`. La seconde formule, pour les cellules `L3C2` à `L13C2`, multiplie la cellule située à gauche, indiquée par `LC(-1)` à celle située au-dessus (`L(-1)C`), ce qui donne la formule récursive classique $fact(n) = n.fact(n-1)$ employée dans la plupart des langages de programmation.

```
1  =L(-1)C+1
2  =L(-1)C*LC(-1)
```

Un tableur est donc capable de reproduire les trois instructions fondamentales

	1	2
1	Somme	Factorielle
2	1	1
3	2	2
4	3	6
5	4	24
6	5	120
7	6	720
8	7	5 040
9	8	40 320
10	9	362 880
11	10	3 628 800
12	11	39 916 800
13	12	479 001 600

FIGURE 2.4 – Tableur calculant la fonction factorielle de manière itérative

de tout langage de programmation, outre les expressions arithmétiques : l'assignation de valeurs, le branchement conditionnel, et l'itération. Avec ces instructions, il est possible de construire une machine de Turing au sein d'un tableur. Celui-ci est donc bien un langage de programmation Turing-complet*, comme cela a été déjà plusieurs fois noté[81].

Une autre direction dans laquelle s'est engagée la démocratisation de la programmation sont les langages visuels, qui ont une longue histoire[82]. Le programme est créé par manipulation d'éléments visuels plutôt que par saisie de texte.

La figure 2.5 représente l'environnement de développement Scratch, un langage de programmation populaire d'apprentissage pour les collégiens, développé par le MIT[83]. Cette vue est découpée en quatre zones. Dans la zone A se trouve l'exécution du jeu « Pong », consistant pour le joueur contrôlant la petite raquette verte à faire rebondir la balle (jaune) avant qu'elle touche la ligne rouge située en bas de la zone. Les trois autres zones sont destinées au programmeur. La zone B répertorie les objets actifs dans le jeu ainsi que leurs paramètres, à savoir la raquette, la balle et la ligne rouge. La zone C permet d'inspecter le code associé à chacun de ces objets. Dans cette figure on voit le code associé à la balle, qui est en l'occurence le plus important. Le code qui est associé à la raquette spécifie seulement comment celle-ci doit se déplacer avec le contrôle de la souris, et aucun

81. Voir par exemple (GREEN 1990a, p. 36)
82. Voir (MARTINI 2021), et (KELLEHER et PAUSCH 2005, p. 91-98) pour une revue historique.
83. scratch.mit.edu.

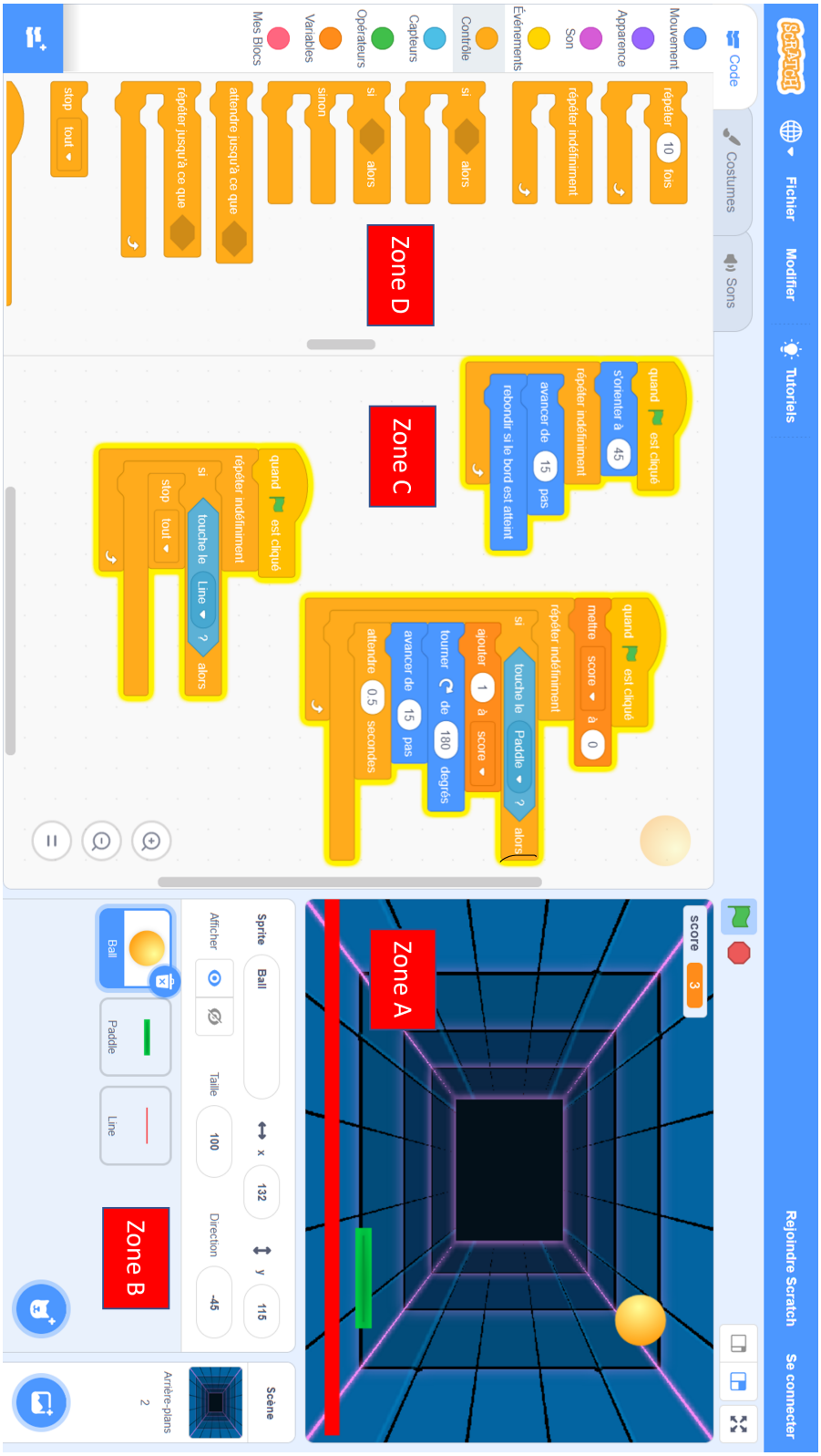


FIGURE 2.5 – Vue du programme « Pong » dans l'environnement de développement Scratch
Source : scratch.mit.edu

code n'est associé à la ligne rouge. Si on inspecte le contenu de la zone C, on peut observer qu'il comprend trois blocs, chacun constitué de plusieurs briques de différentes couleurs s'encastrant les unes dans les autres. C'est le programmeur qui arrange ces briques entre elles, en allant les chercher dans la zone D qui constitue sa « boîte à outils » en quelque sorte. Scratch est ainsi le prototype des langages de programmation visuels.

Les trois blocs de la zone B contrôlent respectivement, dans le sens de la lecture, le mouvement générale de la balle, le rebond de la balle sur la raquette, et enfin l'interaction de la balle avec la ligne rouge, qui termine la partie. La lecture est assez intuitive et se passe d'explications. On voit que Scratch est un langage de programmation impérative* très classique, qui présente des branchements comme `Si touche le Paddle? alors` (bloc 2) ou des boucles comme `répéter indéfiniment`, présente dans les trois blocs.

Dans le domaine des usages de gestion, la programmation visuelle a donné naissance depuis quelques années au développement *no-code / low-code* (*sans-code, peu-de-code*) qui vise à mettre à disposition d'utilisateurs avancés des outils pour concevoir des applications simples, qui permettent en général d'automatiser des flux de tâches* connectés à des bases de données ou à d'autres applications. Par exemple, dans la figure 2.6, l'automatisation des tâches numériques d'un processus de recrutement est organisée. L'outil permet de concevoir le flux de tâches* complet, qui entrelace des activités menées par des personnes, comme l'entretien de recrutement, et des tâches électroniques comme l'envoi de messages électroniques aux candidats. L'application ainsi conçue permet de planifier les activités que le recruteur doit réaliser, de le décharger des tâches électroniques automatisables, et de disposer également d'un tableau de bord général sur l'activité de recrutement, sans qu'il y ait eu besoin de programmer au sens où on l'entend habituellement.

Il est particulièrement intéressant d'explorer ces tendances simplificatrices de la programmation dans le domaine des jeux. Scratch tente de rendre la programmation ludique non seulement par son interface visuelle, mais aussi par son catalogue d'instructions particulièrement adapté à la construction de jeux, ou plus généralement d'animations. On a vu en effet qu'il permet la manipulation d'objets animés sur l'écran grâce à des instructions comme `tourner`, `avancer`, `rebondir`, `touche` par exemple.

À ce titre, on peut rapprocher Scratch de certains jeux numériques qui permettent au joueur passionné de créer de nouveaux « niveaux » de jeu. Les jeu

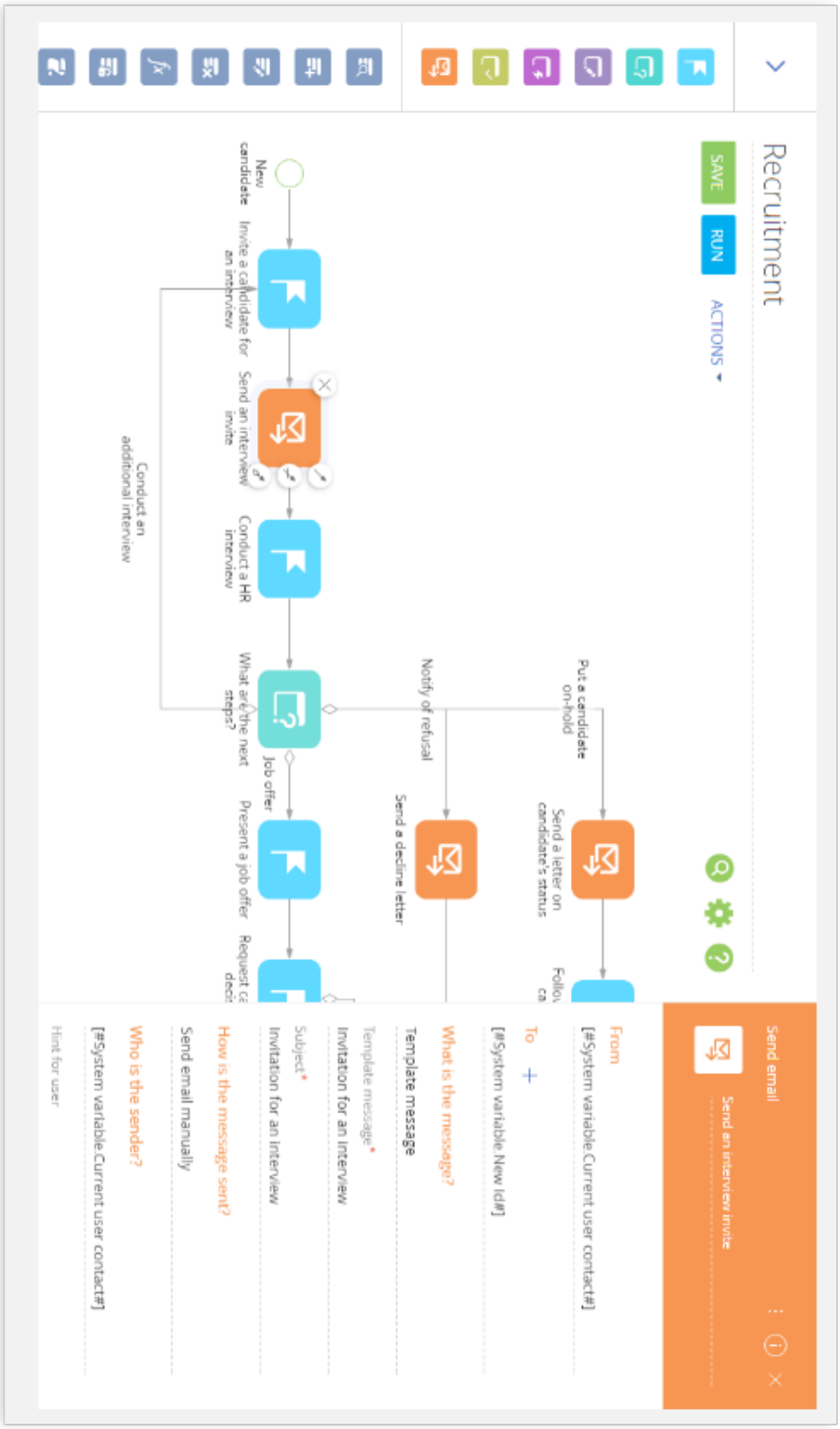


FIGURE 2.6 – Construction d'une application de recrutement sur la plateforme Creatio
Source : www.creatio.com

d'arcades* ont très tôt offert cette possibilité, peu après leur apparition dans les années 1980. Un éditeur de niveau permet, de manière assez classique, de créer un nouveau parcours sur l'écran, de placer des embûches et des ennemis, de contrôler leur comportement grâce à une série de propriétés qui leur sont associées, de définir des conditions diverses qui permettent de progresser dans le niveau – comme par exemple, une porte qui ne peut s'ouvrir que si le joueur en possède la clé, etc.

Certains éditeurs permettent de créer de nouveaux objets et de programmer leur comportement grâce à un langage de scripts* spécial, comme Lua qui est très répandu pour ces usages. On rejoint ici la programmation à part entière, bien qu'elle soit contrainte au contexte donné par le logiciel. Il y a d'ailleurs, dans les plateformes de développement de jeu récentes comme Unreal, de moins en moins de différences entre le langage de scripting* et le langage de développement du jeu lui-même[84].

Il existe d'autres usages du scripting* au sein des jeux. Il permet notamment d'automatiser les mouvements du personnage contrôlé par le joueur, de manière à ce que des tâches fastidieuses soient automatisées, ou qu'il réagisse plus vite à des séquences d'action rapides. Cet usage du scripting* est parfois considéré comme déloyal.

Dans d'autres jeux cependant, où la planification est centrale, le scripting* peut être partie intégrante du jeu lui-même. Dans un jeu de stratégie récent appelé Command : Modern Operations[85] par exemple, un joueur montre comment scripter* en Lua le comportement d'une base anti-aérienne pour que, si la tour de contrôle est détruite, et qu'elle ne peut plus donc envoyer ses informations radar aux batteries disséminées autour de la base, ces dernières activent automatiquement leur propre radar afin de pouvoir agir de manière autonome[86].

Ce jeu, qui a été adopté par certaines organisations militaires pour leurs propres usages d'apprentissage et de simulation (une tendance qu'on appelle le *jeu sérieux* (*serious gaming*)) pourrait être assimilé à de la programmation sans même qu'il y ait besoin de scripting* : l'essentiel du jeu, en effet, consiste à *instruire* ses unités de combat à un certain comportement pendant la phase d'engagement. Cette instruction permet non seulement d'assigner des objectifs à ses unités, mais également de les coordonner entre elles, de les faire réagir à des

84. Voir à ce sujet le chapitre scripting du livre (anonyme) en ligne *Level Design Book*.
85. Voir le site matrixgames.com.
86. (Gatcomb 2021).

événements ou à des informations obtenues grâce à des radars, et de déterminer des règles d'engagement quand elles identifient des unités adverses. Cette instruction est d'une telle précision qu'elle semble ne différer du scripting* qu'autant que la programmation visuelle comme Scratch diffère de la programmation textuelle classique.

Cet exemple permet de voir jusqu'à quel point la programmation peut s'infiltrer au sein des usages numériques : dans le cas des jeux, et en particulier des jeux de stratégie et de gestion, cette intégration est d'autant plus fluide que ces jeux *eux-mêmes* semblent n'être rien d'autre que de la programmation en puissance – certes pas d'ordinateurs, mais d'objets complexes qui *agissent comme des machines* et qui peuvent être contrôlés comme elles.

2.5 Une tentative de définition

Après avoir présenté les quatre formes de transformation de la programmation, et illustré en détail deux d'entre elles (la programmation spécialisée et la programmation visuelle ou simplifiée), nous en donnons une définition très générique que nous commentons en détail sur deux développements (§ 39 et § 40). Le développement suivant complète ce commentaire par l'examen de termes voisins de *programme*, comme *procédure*, *code* et *algorithme* (§ 41).

§ 39. La définition (1) : Une visée intentionnelle...

À la fin de l'introduction de ce chapitre[87], nous avions suggéré, contre Welsh, que c'était toute activité de « diriger un ordinateur à faire ce qu'on veut » qu'il fallait appeler *programmation*, et non seulementce qu'il semble entendre par là, qui correspond à peu près à ce qu'on appelle *codage*[88].

Cette acception large est confirmée et précisée par la définition du terme *programme* que propose le groupe de recherche PROGRAMme[89], auquel nous avons participé :

> Un arrangement de signes visant à déterminer le comportement d'une machine, dans une situation donnée[, afin de transformer celle-ci d'une manière également déter-

87. Voir plus haut, § 25.
88. Sur ce terme, voir plus loin § 41.
89. programme.hypotheses.org.

minée][90].

Selon qu'on entend *arrangement* en un sens dynamique (l'action d'arranger) ou statique (le résultat de cette action), cette définition vaut aussi bien pour la programmation que pour le programme. Elle va nous servir de point de référence dans l'ensemble de notre travail, aussi est-il utile de la commenter terme à terme.

1) Signes. Un programme est composé de signes. Il n'est pas spécifié ici de quelle nature sont les signes ; ils peuvent être des sons, des inscriptions sur papier, des trous sur des cartes perforées, des positions d'interrupteurs, des signaux électriques, ou encore des représentations mentales, des gestes du corps, etc. L'important est qu'ils soient *institués* en avance du programme lui-même. Cela est en effet nécessaire afin qu'ils puissent avoir une relation déterminée avec le comportement attendu de la machine. Ils sont donc en nombre fini.
De nombreuses distinctions (qui sont parfois source de difficultés ou de confusions) concernant les signes valent également pour les programmes. D'abord, un signe peut être tout aussi bien un type général qu'un individu concret de ce type (qu'on appelle *token* en anglais), d'être la lettre « a » aussi bien que cette lettre qui se trouve 176 caractères (y compris les espaces) avant le point terminant cette phrase, sur ce papier ou cet écran que vous lisez à l'instant. De la même manière un programme peut donc aussi bien être *cet exemplaire* du programme, qui est en ce moment enregistré sur tel disque, que ce programme comme type général, matérialisé par toutes ses réalisations individuelles, et qui peut toujours (en droit, comme n'importe quel texte) être recopié à l'identique.
Un signe peut être une inscription ou un signal, une distinction dont le modèle est celle entre l'écriture et la parole. Un programme est en général de la première sorte pour deux raisons : d'abord, il y a l'idée qu'il est un objet permanent et réutilisable : un même programme peut servir à plusieurs exécutions sur la même machine ou sur des machines différentes, il peut être recopié, etc. Comme on l'a vu plus haut[91], un programme est souvent destiné à rester actif plusieurs années. Deuxièmement, il y a l'idée qu'il peut être inspecté, analysé, et modifié. Un programme dont le support serait une simple gestuelle devrait être enregistré en vidéo afin que cela soit possible. Ce caractère objectif est sans doute le point central ici. On peut le voir en considérant les interfaces cerveau - machine, qui

90. La version originale de cette définition se trouve chez Mélès (2022, p. 3).
91. Voir plus haut, § 30.

permettent à une personne de contrôler un ordinateur[92]. Si celle-ci prend l'habitude de lui envoyer une certaine suite d'instructions mentales qui déterminent un certain comportement de la machine, de telle manière que cette communication devienne automatique et inconsciente, parlera-t-on encore de *programme* en ce cas ? Cet événement purement privé ne pourrait être considéré comme tel, sans doute, que par un observateur tiers capable d'enregistrer ces impulsions.

2) Arrangement. Plus que les signes eux-mêmes, c'est leur arrangement qui fait le programme. On pourrait aussi utiliser les termes *structure, disposition, organisation, ordonnancement, schéma*. La plupart de ces termes peuvent être aussi entendus en un sens dynamique et donc valoir, comme nous l'avons déjà noté, pour la programmation également.

En écriture binaire, où deux signes seulement sont autorisés, leur arrangement séquentiel *fait* le programme. L'arrangement peut être linéaire, plan, ou encore tri-dimensionnel, temporel, etc. Ainsi il peut prendre une forme graphique par exemple, comme un diagramme de flux*, et il pourrait également utiliser d'autres propriétés de la matière arrangée, comme la couleur, etc.

Les règles de l'arrangement peuvent être instituées en avance du programme, comme le sont les signes (par exemple, que chaque ligne ne peut contenir qu'une seule instruction) – mais cela n'est pas nécessairement le cas : lorsqu'une série de symboles présente une régularité seulement statistique, la règle de leur arrangement peut être *induite* par la machine. Ce point permet de couvrir le cas de la programmation par apprentissage évoquée plus haut.

Il faut ici noter qu'un programme ne doit jamais être confondu avec aucune de ses exécutions particulières, ce que le langage courant encourage parfois, quand on dit que « *le programme demande ceci ou cela à l'utilisateur* » ou que « *le programme donne comme résultat ceci ou cela* ». Le « sujet » des processus informatiques observés, si cette expression a un sens, ne peut être rien d'autre que la machine.

3) Visant à. L'arrangement est destiné à déterminer le comportement de la machine, et par ce biais, à transfomer une situation globale. Un programme est donc un objet intentionnel, qui suppose un sujet de cette intention, le programmeur. Le programmeur n'est pas forcément le codeur du texte, ni même le concepteur du programme. Il est souvent constitué d'une équipe entière, qui comprend un client définissant ses attentes, un architecte, des codeurs, des testeurs, etc. Des machines (par exemple des compilateurs) peuvent participer à l'effort de programmation.

92. (Berger et Mattout 2017).

Il n'est pas dit ici que cette intention va être effective. Le programmeur peut se tromper dans sa visée, et mal évaluer la chaîne causale qui permet à des arrangements de signes d'être effectifs dans la situation donnée. En ce sens, comme le remarque FETZER (1988, p. 1062), tout programme est une conjecture que chacune de ses exécutions peut réfuter.

On remarque également que l'intention du programmeur est double, puisqu'elle vise à la fois la modification du comportement de la machine et de la situation. Nous revenons sur cette dualité plus loin.

4) Déterminer. La possibilité que des signes déterminent le comportement d'une machine n'est pas établie à ce stade, et c'est pour cette raison principale que la définition que nous donnons ici est nominale : elle ne démontre pas la possibilité de son objet. Ce problème n'est rien d'autre que celui de l'effectivité des ordinateurs, l'une de nos questions directrices.

D'un point de vue empirique, cette chaîne causale peut être longue. Si par exemple le programme est un diagramme de flux* tracé sur un papier, il doit ensuite être codé dans un langage de programmation, puis compilé et exécuté avant de déterminer le comportement de la machine. On peut imaginer (si on suit Welsh) que l'ensemble de ces tâches soient exécutées par la machine elle-même, y compris l'étape du codage. La chaîne causale peut aussi être directement mécanique, si par exemple le programme est un arrangement d'interrupteurs contrôlant directement le comportement de circuits électroniques. Elle pourrait aussi être de l'ordre de la communication, si on accepte que des personnes puissent jouer le rôle de machines (voir plus loin), et *obéir* au programme en appliquant les règles instituées par avance.

Déterminer est d'ailleurs un terme trop fort, car il y a des programmes non-déterministes, c'est-à-dire dont la visée autorise plusieurs, voire une infinité de comportements de la machine. L'expression exacte qu'il faut employer est : *contraindre, de manière déterminée,* le comportement de la machine.

5) Comportement. Cette dernière formulation met en évidence que tout comportement doit pouvoir être identifié comme élément d'un espace de comportements possibles, qu'on sait parcourir descriptivement (désigner des classes de comportement, distinguer un comportement d'un autre, etc.).

Cela est possible car un comportement n'a de sens que relativement à un observateur. Le sens moderne de ce terme provient en effet de l'éthologie et de la psychologie, notamment des behavioristes qui souhaitaient décrire scientifiquement

les processus signifiants chez les humains et les animaux sans aucune référence à des entités inobservables telles que les pensées ou les émotions. Ils voulaient aussi éviter de réduire ces processus à de purs mouvements physiologiques ou neurologique[93]. *Comportement* fait donc référence à un niveau intermédiaire de description, que les expérimentations du scientifique dégagent minutieusement des régularités observées de l'organisme soumis à certaines conditions contrôlées. Dans le schéma le plus courant, stimulus - réponse, c'est le scientifique-observateur qui délimite ce qui doit compter comme tel. Il est à sa charge de prouver que des régularités intéressantes peuvent être observées à partir de cette sélection[95].
En informatique, le schéma équivalent est « entrée - sortie ». Les observables peuvent être par exemple des résultats qui s'affichent à l'écran ou sur une imprimante, la mise à jour d'un fichier en mémoire, la fermeture d'une valve dans une écluse.
Le comportement peut cependant être défini de manière plus large, et consister dans une série temporelle d'observables, par exemple la cinématique d'un graphique qui se déroule sur l'écran, ou le dialogue complet de la machine avec ses utilisateurs, les mises à jour *dans le temps* d'une base de données, etc. De nombreux programmes en effet n'ont pas vocation à se terminer et leur comportement ne peut être donc défini par leur résultat final, mais bien plutôt par le flux des observations faites concernant la machine durant l'exécution du programme.
L'observateur primaire est le *programmeur*, car c'est lui qui vise à établir la relation signifiante entre les signes du programme et le comportement attendu de la machine. Lui seul peut vraiment *observer* la machine, car lui seul sait *ce qu'il faut regarder*. Le programmeur peut définir d'autres rôles d'observateurs, qui n'ont accès qu'à des sous-ensembles d'observables. On les appelle en général *utilisateurs*. Ils peuvent également avoir des attentes différentes de celles du programmeur concernant le comportement du programme. Par exemple, les utilisateurs du moteur de recherche Google s'attendent uniquement à ce que ce programme fournisse des résultats pertinents, alors que l'entreprise Alphabet (propriétaire de Google) s'attend également à ce qu'il diffuse des publicités et collecte des données sur les utilisateurs.

=

93. Voir par exemple [94].
95. (Skinner 1938, p. 9).

§ 40. La définition (2) : ... Portant sur des machines en situation

Nous continuons le commentaire de notre définition :

6) Machine. La définition des observables implique qu'il faut définir les frontières de la machine, pour la distinguer de son environnement externe. Par exemple on peut, dans le cas d'un programme devant produire un texte imprimé, définir l'observable comme le texte imprimé sur une feuille de papier, ou uniquement comme le fichier postscript envoyé à l'imprimante. Dans le premier cas, l'imprimante fait partie de la machine, dans le second, elle n'en fait pas partie. L'observateur pourrait définir la machine encore plus étroitement, par exemple en la limitant au seul processeur de l'ordinateur, si le programme concerne du code assembleur*. Dans ce cas, les observables peuvent être limitées aux valeurs des registres* centraux du processeur et à ses flux d'entrées et sorties avec d'autres composants de l'ordinateur. Inversement, l'observateur peut définir la machine comme un réseau complet d'ordinateurs, par exemple lorsque le programme concerne l'orchestration de services distribués.

Les frontières de la machine ne sont pas seulement liées à son environnement externe. La couche physique des composants de la machine, bien qu'« à l'intérieur » de la machine, peut également être considérée, lorsque cela convient à l'observateur, comme hors de son périmètre. Par exemple, certains composants peuvent s'occuper de leurs opérations de niveau inférieur de manière assez autonome et envoyer des messages ou des alertes réguliers à la machine principale. Ou si l'on reprend l'exemple du réseau informatique, il peut être dans l'intérêt du programme d'orchestration de considérer les ordinateurs individuels comme ayant un comportement autonome, et donc comme hors de ses frontières, et de se focaliser sur le comportement de leurs seules interactions.

Jusqu'ici nous avons pris *machine* dans le sens que le contexte de ce travail lui donne naturellement, celui d'un ordinateur électronique. Cependant, la définition de *programme* continue de valoir si on prend *machine* dans un sens plus large.

D'abord, on parle en informatique théorique de *machine abstraite* ou *logique* pour désigner (en général, et de manière simplifiée) des structures mathématiques dotées d'un espace d'*états* et d'un endomorphisme sur cet espace, appelé *fonction de transition*. Un programme est alors un ensemble de règles qui décrivent le comportement de cette fonction, comme par exemple la table d'instructions d'une machine de Turing. Notre définition continue de s'appliquer. Cette remarque est importante, car elle montre la *neutralité ontologique* des concepts dont on s'oc-

cupe ici. Un programme n'est pas concret ou abstrait, physique ou mental, particulier ou général : il a le même statut que la machine dont traite le discours.
De nombreuses machines sans électronique peuvent également être programmées. Pour prendre un exemple du quotidien, on peut « programmer » un lave-vaisselle en tournant quelques boutons sur son panneau de commande : ces boutons sont ici les « signes » de notre définition. Cela est encore plus vrai des machines industrielles, qui sont d'ailleurs de plus en plus souvent paramétrées par ordinateur – nous donnerons plus tard dans ce travail l'exemple du taillage de verres ophtalmiques[96]. La définition continue de valoir à travers les nombreux sens métaphoriques dans lesquels on peut prendre *machine*. Par exemple, une partition est un « programme » pour la « machine » qu'est l'interprète ou l'orchestre, de même qu'une procédure pour une administration ou une organisation. Il semble ainsi que, dès qu'est présente l'idée de machine, même métaphoriquement, et sans qu'on pense nécessairement à l'ordinateur, celle de programme est également possible. Une machine peut certes se réduire à un simple mécanisme absolument rigide ; mais le plus souvent, on entrevoit l'idée qu'un mécanisme peut être configuré, c'est-à-dire programmé[97].
7) Dans une situation donnée. Ce complément circonstantiel peut paraître superflu. N'est-il pas évident que tout comportement est situé et dépend de cette situation ? Ce qu'on cherche à souligner ici, c'est que la situation *fait partie* de la visée du programmeur, c'est-à-dire qu'il ne peut attendre de la machine qu'elle se comporte de telle ou telle manière qu'en ayant à l'esprit, même confusément, une série de conditions externes à ce comportement. On peut les classer en deux types. Les *conditions matérielles* portent sur le fonctionnement de la machine elle-même : qu'elle ait de l'électricité par exemple et qu'elle soit bien conçue – notamment qu'elle « comprenne » les règles et les signes avec lesquels le programmeur élabore son programme. Les *conditions d'usage* concernent le comportement attendu de la situation (notamment de ses utilisateurs) afin que la machine puisse exécuter correctement le programme. À un niveau élémentaire, il peut s'agir des conditions d'interaction entre le monde externe et la machine, par exemple, un certain format peut être requis pour les données entrées par l'utilisateur. Cependant, des conditions de comportement plus profondes peuvent être requises[98].

96. Voir plus loin § 208.
97. Nous revenons longuement sur ces points au chapitre 11.
98. Nous en verrons des exemples plus loin, par exemple § 51.

L'informaticien Michael Jackson distingue trois classes de conditions d'usage[99]. Il peut y avoir des conditions causales, ancrées dans les lois naturelles – par exemple, un programmeur dont la visée est de contrôler une écluse présuppose que ses vantaux obéiront aux signaux que l'ordinateur leur enverra. Il peut y avoir des conditions concernant les conduites humaines, notamment celles des utilisateurs du programme. Il peut y avoir des conditions symboliques, qui portent sur les données reçues par la machine (qui peuvent provenir de personnes humaines ou d'autres machines).

En programmation fonctionnelle*, où les données d'entrée du programme peuvent uniquement être fournies à l'ordinateur au lancement de l'exécution, les conditions d'usage sont entièrement concentrées dans cette unique interaction initiale. Après cela, la machine exécute le programme sans que l'environnement externe puisse intervenir sur son cours. On peut imaginer enfin des programmes complètement « fermés » au monde externe, qui exécutent une routine fixe, fermée à toute paramétrisation par l'utilisateur. Dans ce cas la situation est réduite à ses seules conditions matérielles.

8) Afin de transformer la situation. Cette dernière partie de la définition est mise en apposition et entre crochets, afin de signifier son caractère optionnel. Elle ne figure pas d'ailleurs dans la définition retenue par le groupe de recherche PROGRAMme.

Nous l'avons cependant ajoutée pour deux raisons. Elle est d'abord conforme au sens commun. Un programme, ainsi que la machine qui l'exécute, sont en général perçus comme ayant un caractère utilitaire – ils appartiennent au monde de la technique – même si cette utilité sert des fins théoriques, par exemple mathématiques ou scientifiques. Ensuite, c'est bien de ce point de vue que nous souhaitons dans ce travail étudier les programmes et la programmation, puisque nous nous intéressons à l'*effectivité* des ordinateurs.

Cependant, d'un autre point de vue, cette apposition n'est pas nécessaire à la définition stricte de ce qu'est un programme. Les jeux évoqués plus haut n'ont pas de caractère utilitaire. On pourrait répondre ici qu'ils modifient bien la situation dans laquelle se trouve l'utilisateur qui, en lançant un jeu, se met à jouer et à éprouver les sentiments associés à cet état – ou en tous les cas, telle est bien la visée du programmeur de jeux.

C'est pour étendre l'acception de notre définition à des cas comme ceux-ci que

99. (M. Jackson 2005, p. 907).

nous n'employons pas ici le vocabulaire de la résolution de problème – où la situation de départ serait le problème à résoudre, et la situation transformée sa solution. Ces expressions connotent d'ailleurs trop des situations statiques de type « entrée - sortie », alors que la transformation d'une situation peut dénoter également la modification de processus dynamiques, par exemple le passage d'un travail manuel à un travail automatisé. Par ailleurs, même dans le cadre d'une résolution de problème, la programmation peut être simplement une étape de l'investigation, et non se substituer à elle, comme lorsqu'on utilise un ordinateur pour explorer certains scénarios ou réaliser des calculs intermédiaires. Enfin, elle peut consister à construire des outils ou des plateformes génériques de résolution de problèmes, comme lorsqu'on programme un langage de programmation. Dans tous ces cas de figure, on peut dire que le programme transforme la situation problématique, mais il ne la résout pas. On verra d'ailleurs plus loin[100] que *situation problématique* nous semble être un terme plus fondamental que *problème*.

On comprend donc qu'il faut entendre la *situation* dans le sens le plus large possible. Elle est ontologiquement neutre. On peut la définir par des suites de symboles, des configurations matérielles, ou des états psychologiques, et elle peut être statique (un objet ou une composition d'objets) ou dynamique (un processus ou un ensemble de processus). On peut même définir la situation comme une conjecture mathématique en cours d'examen, où le programme sert à tester certains cas de figure[101]. Cette neutralité s'explique par ce que dans notre définition, tout comme la machine, la situation est partie de la visée du programmeur. Rien n'est affirmé quant à « ce qui existe vraiment », mais seulement quant à la manière dont le programmeur se représente le problème qu'il a à résoudre.

Ces formulations se révèlent néanmoins insuffisantes à rendre compte d'autres types de programmes, notamment ceux qui résultent d'un projet de création artistique. On peut concevoir en effet des programmes qui n'ont pas d'autre visée que leur propre exécution. Il se pourrait par exemple que, à la manière des *ready-made* de Duchamp, le geste artistique soit simplement d'*ôter* sa fonction au comportement de la machine ; il serait incorrect alors de les qualifier d'*outils*. Par exemple, le *Glitch Art* vise explicitement à détourner des algorithmes de compression afin de créer des animations numériques dysfonctionnelles[102]. C'est pour cette raison

100. Voir plus loin, § 167.
101. Nous revenons sur ce type de situation vers la fin de ce travail, § 195.
102. (C. Morgan 2019).

qu'il est prudent ici d'affirmer le caractère *optionnel* de cette finalité seconde de la programmation. Dans la perspective que nous avons esquissée en introduction, cependant, une telle réserve n'a pas d'importance[103].

Au terme de ce commentaire, il apparaît que la définition dessine un arrière-plan riche de structures descriptives des phénomènes de la programmation, qu'on peut présenter par un faisceau de polarités :

1) Programme / machine. Le programme décrit le comportement de la machine, il est donc distinct d'elle. En anglais, cela donne la distinction célèbre entre *software* et *hardware*, entre logiciel et matériel informatique.
2) Programmeur / machine. La machine est la chose programmée (passive) et le programmeur est le sujet intentionnel (actif). Le programmeur est si bien distingué de la machine qu'il reste implicite dans la définition.
3) Programmeur / utilisateur. Le programmeur a, selon la définition, accès à l'ensemble des observables (puisque c'est lui qui les construit), tandis que l'utilisateur n'a accès qu'à ceux accordés par le programmeur. L'utilisateur est un rôle *créé* par le programmeur. Dire que l'utilisateur, tout en se limitant au rôle attribué par le programmeur, peut néanmoins programmer également, suggère une mise en abyme possible de cette relation.
4) Programme / données. En particulier, l'utilisateur (et plus généralement la situation) peut seulement fournir *des données* à la machine au début ou au cours de l'exécution du programme. Par opposition, tout ce que le programmeur fixe en amont de l'exécution (y compris les bases de données qui, parfois, alimentent le processus informatique) sont partie du programme, et non des données.
5) Machine / situation. Comme nous l'avons vu, la machine doit être stipulativement distinguée de la situation, afin qu'on puisse savoir *quoi observer* de son comportement.
6) Visée / comportement. L'intention du programmeur est distincte du comportement réel de la machine, qui peut la décevoir.
7) Programmation / exécution. Deux temps sont fortement distingués : celui de l'arrangement des signes (la programmation) et celui de l'exécution du programme. Les sépare un moment central, qui est celui du lancement de l'exécution (*run-moment*). Ce moment structure l'organisation même du travail informatique, entre développement et supervision des logiciels.

103. Voir plus haut, § 11.

§ 41. Plan, procédure, algorithme

Afin de compléter notre définition du terme *programme*, il est utile de la contraster avec celles de quelques notions proches, souvent tenues pour synonymes. Nous développons en particulier celles de *plan, procédure, algorithme* et, plus rapidement celles de *code, script, routine, application, logiciel*.

Plan est étroitement associé à *programme* aux débuts de l'informatique. Priestley (2017, p. 459) montre que les premiers informaticiens conçurent les premières méthodes de programmation sur le modèle de la planification des calculs de grande ampleur qui avaient lieu dans les bureaux de calculateurs (humains), devenus courants au début du xx^{e} siècle[104]. Grace Hopper, une informaticienne célèbre à l'origine du langage COBOL, propose la définition suivante en 1954 dans le premier glossaire de l'ACM :

> *Programme*. – Un plan pour la résolution d'un problème. Un programme complet inclut des plans pour la transcription des données, le codage de l'ordinateur et des plans pour l'absorption (sic : *absorption* en anglais) des résultats dans le système. La liste des instructions codées est appelée une routine[105].

Ici, le programme apparaît avant tout comme une méthode pour résoudre un problème, et c'est « la liste des instructions codées » qui correspond strictement à notre définition. Cependant, Hopper admet qu'en un sens (« complet ») le codage final fait également partie du programme. Ce hiatus témoigne d'une évolution progressive du terme vers sa signification actuelle. Comme l'explique le collectif PROGRAMme,

> À l'époque, la programmation était souvent décrite comme « la planification et le codage d'un problème », la première faisant référence à l'activité soi-disant mathématique de la construction du plan de calcul, et le second à la tâche soi-disant administrative (*clerical*) de son encodage sous la forme d'instructions pour la machine[106].

On ne concevait donc pas encore à cette époque, qu'il put y avoir un objet unique à double visée – celle d'un certain comportement de la machine *et* de la résolution d'un problème. Au contraire, *coder* était justement l'action de transformer des significations visant à résoudre un problème (le plan) en d'autres visant le comportement de la machine (la routine). La question qui nous occupe donc, celle du

104. Voir également plus loin notre développement à ce sujet § 85.
105. Cité par (Lonati, Brodnik et al. 2022, p. 3).
106. Collectif PROGRAMme, *What is a computer program ?*, à paraître.

mystère de cette transformation, est donc laissée complètement béante. Il n'est pas dit par exemple quels types de plans humains peuvent s'y prêter. Nous revenons sur la notion de *plan* au chapitre 5, § 85.

Code et *routine* doivent donc être compris, au moins dans leur sens historique originel, comme les compléments du *plan*. Ils évoquent le texte même du programme, indépendamment de son intention ou de sa spécification. La définition de Hopper suggère une *certaine forme* restreinte des signes et de leurs arrangements, celle d'une séquence d'instructions énoncées sur le mode impératif, dont nous avons vu qu'elle devait être relâchée, par exemple en référence à la programmation fonctionnelle* et à la programmation visuelle.

Code s'est connoté positivement depuis quelques années avec la montée en puissance des méthodes agiles*, qui favorisent l'écriture directe de programmes[107]. Comme le notent De Mol et Bullynck (2022, p. 10), le terme évoque à présent le *hacking* et les aspects récréatifs et souvent virtuoses de la programmation. *Routine*, au contraire, est passée de mode, et évoque de ce fait les premiers temps de l'informatique, ceux des *mainframes**, et dans le meilleur de cas, l'exécution de processus informatiques simples, en général fonctionnels (à entrée - sortie), servant de chevilles à des programmes plus complexes.

Procédure est un terme à géométrie variable, qui le rend très utile à notre réflexion. Comme nous l'avons rappelé en introduction[108], il s'est doté au début des années 1930 d'un sens logique (*procédure effective*) qui le rend significatif en informatique théorique ; il se rapproche alors d'*algorithme*. Mais il est également utilisé par les praticiens, pour qui il est souvent synonyme de « sous-programme » : ce sens est consacré par le langage Pascal qui en fait un des termes de sa syntaxe. Cependant le terme garde aussi, bien plus que *programme*, sa pertinence dans des domaines extra-informatiques, notamment le droit, l'administration, le travail technique. Il est donc aussi proche de *plan*, en ce qu'il ne fait pas référence à la machine, et qu'il a une forte connotation impérative.

Algorithme a pris son sens moderne au XIX^e^ siècle en mathématiques, où il est très proche de *méthode*, terme qui reste à cette époque de loin le plus courant[109]. Cependant, son usage ne se développe qu'à la fin des années 1950, grâce à la référence qui lui est faite dans l'acronyme du langage de programmation célèbre

107. Voir plus haut, § 34.
108. Voir plus haut, § 13.
109. (De Mol et Bullynck 2022, p. 4).

ALGOL. Comme le remarque Naibo, cette perspective a eu pour conséquence de restreindre l'idée d'algorithme à celle d'une procédure effective de traitement symbolique, et il suggère à bon droit qu'il faut, pour bien la comprendre, l'envisager d'une manière générale comme « méthode de résolution de problèmes permettant l'acquisition d'une connaissance[110] », ou encore « un ensemble structuré d'opérations qui sont censées agir directement sur des entités (mathématiques) abstraites et qui sont indépendantes de la manière dont elles sont implémentées (spécifiées) afin d'être exécutées[111] ». Le programme, dans cette perspective, est justement ce qui implémente un algorithme et permet de l'exécuter. Dans la définition de Hopper, l'algorithme correspond donc au *plan*, et le programme à son codage en une routine.

Un programme implémente donc un algorithme comme un décret implémente une loi. En particulier, un programme *traduit* également l'algorithme dans un langage particulier. Knuth (1974, p. 323), qui fait des algorithmes l'objet de l'informatique comme science, dit que « la représentation particulière d'un algorithme est appelée un *programme*, tout comme nous utilisons le mot *donnée* pour signifier une représentation particulière de l'*information* ». Cela suppose qu'on puisse décrire les algorithmes comme des objets mathématiques à part entière et les étudier comme tels indépendamment de toute « représentation particulière ». Or ce que signifie une telle expression reste une question ouverte, qui occupe une place importante actuellement en philosophie de l'informatique et des mathématiques[112].

En pratique, on étudie souvent les algorithmes et leurs propriétés dans le modèle de référence des machines de Turing. Cependant ce modèle n'en est qu'un parmi d'autres possibles, comme le λ-calcul* ou les système de production*, qui ont une puissance calculatoire équivalente. Or un « même » algorithme (par exemple celui qui donne la suite de Fibonacci) est exprimé de manière différente dans chacun de ces modèles de référence, de telle sorte que le critère d'identité qui permet d'affirmer qu'il s'agit bien du « même » calcul devient difficile à préciser[113]. Plusieurs propositions ont été émises pour trouver un modèle de référence mathématique assez abstrait pour pouvoir rendre compte de tous les autres, dont une récente est celle de Naibo (2022). Il est important pour notre propos que la plupart de ces modèles se fondent sur l'idée de machine abstraite, en incorporant de

110. (Naibo et Seiller 2022, p. 35).
111. (ibid., p. 10).
112. (Blass, Dershowitz et Gurevich 2009, p. 145).
113. (Naibo 2022, p. 8-9).

manière plus ou moins directe les notions d'*état* et d'*action* (ainsi les machines de Kolmogorov-Uspensky, les machines à état abstrait de Gurevich, les espaces et les actions de Seiller). Il semble donc que, comme la notion de programme, et contrairement au plan ou à la procédure, la notion d'algorithme requiert bien nativement celle de *machine*, ce qui en marque le caractère distinctement informatique.

Une autre stratégie, proposée par Robin Milner[114], est de définir les algorithmes comme classes d'équivalence de programmes écrits en différents langages ou modèles. Cependant, il ne semble pas possible de définir précisément quelle serait cette relation d'équivalence qui permettrait de passer de l'expression d'un algorithme dans le langage A à une autre dans le langage B. On peut toujours concevoir un compilateur qui traduit un programme d'un langage à l'autre, mais le résultat n'aura sans doute pas grand chose à voir avec l'expression naturelle de l'algorithme dans le langage cible. L'équivalence entre deux expressions semble chaque fois dépendre de l'algorithme considéré, et de nos justifications intuitives de pourquoi on a bien affaire, *dans ce cas-ci*, au *même* algorithme[115].

Étant donné ces difficultés, certains, comme Dean (2016) ou Bergstra (2021), rejettent l'idée que les algorithmes soient des objets théoriques à part entière. Nous n'en aurions qu'une compréhension pratique, dans le cadre de leur usage pour la résolution de problème.

Comme nous l'avons dit en introduction, notre angle d'attaque est l'acte de programmer lui-même, et non son résultat, le programme. Or l'acte de programmer contient, dans sa dimension réflexive, une part de résolution de problème, et celle-ci exige l'utilisation d'algorithmes, voire leur élaboration. C'est donc bien à partir de cette perspective pratique de la conception et de l'usage des algorithmes, suggérée par Dean et Bergstra, que nous touchons à la question de l'algorithmique. On peut se demander, notamment, si l'essentiel du contenu épistémique de la programmation ne résiderait pas dans cette activité algorithmique – puisqu'après tout, pour reprendre la définition de Naibo plus haut, un algorithme est toute méthode de résolution *d'un problème de connaissance*. Cette perspective peut, selon nous, éclairer la question de la possibilité ou non d'une théorie mathématique des algorithmes, en se demandant ce que voudrait dire de *théoriser* l'activité de résolution de problèmes.

Nous passons plus rapidement sur trois derniers termes situés dans le champ

114. Cité par (Dean 2016, p. 43).
115. (ibid., p. 44-51). Voir aussi (Blass, Dershowitz et Gurevich 2009).

lexical proche de *programme*. Le scripting* est une forme de programmation de haut niveau, visant à diriger l'exécution de programmes dans des contextes particuliers, souvent internes à l'exécution de certains logiciels : on en a vu des exemples dans le cas des jeux[116]. *Application* (ou *app*, plus récemment), et *logiciel* sont des termes évoquant les programmes comme des artefacts conçus en vue d'un usage précis. Ils sont tournés vers des utilisateurs ou des clients qu'ils doivent satisfaire ; c'est donc la situation qui est ici mise en relief. Comme on l'a vu *développement* est le terme consacré pour désigner la programmation professionnelle ; il est emprunté au vocabulaire de l'ingénierie.

Tous ces termes que nous avons passés en revue tendent à confirmer la robustesse de la définition proposée de *programme* : chacun évoque en effet l'un ou l'autre de ses moments – l'arrangement de signes, l'exécution, la machine, la situation, l'usage. *Procédure* et *algorithme* se détachent nettement des autres termes par leur intérêt conceptuel : le premier pose la question de la relation de similarité entre programmes et procédures extra-informatiques (juridiques, administratives, etc.), et le second celle de leur nature mathématique.

2.6 Discussion et critiques

Dans cette section, nous allons mettre à l'épreuve notre définition, en la comparant d'abord à d'autres propositions (§ 42), avant de revenir aux formes extrêmes de programmation envisagées en section 2.4 pour voir si elle permet d'en rendre compte (§ 43). La conclusion est mitigée : si notre définition convient littéralement, elle semble également ouvrir trop grand le spectre de ce qu'est la programmation. Nous concluons (§ 44) qu'elle ne peut être exactement délimitée et distinguée des autres formes de l'usage de l'ordinateur que par le procès épistémique qu'elle implique, et qui la fait apparaître comme une forme très particulière de *réflexion*.

§ 42. Comparaison avec d'autres définitions

Nous nous appuyons ici sur une revue impressionnante des tentatives de définition de la notion de *programme* récemment réalisée [117]. Les auteurs insistent sur la difficulté de cette entreprise, qui est liée selon eux à la nature multi-dimensionnelle

116. Voir plus haut, § 38.
117. (Lonati, Brodnik et al. 2022).

du concept, une définition univoque risquant d'omettre certains de ses aspects.

Ainsi ils remarquent que de nombreuses définitions prennent acte immédiatement de la nature multiple (souvent trinitaire) d'un programme[118]. La définition de Ray Turner par exemple, sur laquelle nous revenons au chapitre 3, décrit le programme comme le *paquetage* (*bundle*) d'une spécification, d'un texte symbolique, et d'un processus physique. Il n'est pas difficile de retrouver ces termes dans notre définition : la spécification est la visée du programmeur, le processus physique est le comportement de la machine, et le texte symbolique, l'arrangement de signes. L'ontologie des programmes proposée par Lando et al. (2007) reprend cette idée, en faisant dériver le concept de *programme* à la fois de celui de *document* et d'*artefact*, ce dernier étant le soutien d'une *action* (ici, le processus de la machine).

Étudiant les diverses définitions collectées dans leur corpus très complet, Lonati, Brodnik et al. (2022) concluent que les dimensions qu'elles mettent en avant ne se recoupent que partiellement. Ils proposent, en guise de synthèse, un *cadre d'analyse* (*framework*) de la notion en six aspects, qu'ils contrastent deux à deux. Les programmes sont donc :

1. Des notations et aussi des entités exécutables.
2. Des réalisations humaines et aussi des outils.
3. Des entités abstraites et aussi des objets physiques.

Il n'y a pas de commentaire spécial à faire sur la première paire d'aspects, qui reprend deux des trois termes de la définition de Turner. La nôtre met de plus en avant le fait que le troisième terme, ici omis, la spécification ou l'intention, est ce qui lie logiquement les notations à l'exécution du programme.

La seconde paire d'aspects envisagent les programmes comme des artefacts, qui sont à la fois des produits et des outils. Ce sur quoi les auteurs cherchent à mettre l'accent ici est est la dimension humaine des artefacts, en particulier leur caractère culturel et social, contre les interprétations qui en ferait des objets a-temporels, produits d'une logique ou d'une technique pure. Mais ce faisant, ils sont confrontés à la très grande part d'automatisation qui caractérise la programmation moderne – par exemple la compilation. Leur commentaire à ce sujet est éclairant :

> Cela ne veut pas dire que le produit final est entièrement fait à la main (*hand-made*), puisque l'automatisation joue un rôle important dans le processus, qui utilise des

118. (ibid., p. 9).

> compilateurs, interpréteurs, et bien d'autres outils, mais à la fin, le programme est le résultat d'une décision humaine de créer quelque chose[119].

C'est donc bien l'intention initiale qui fait qu'un texte symbolique peut être reconnu comme un *programme*, et pour autant qu'une intention ne peut être qu'humaine, la propriété proposée est bien valide. Le mérite de préférer la propriété d'intentionnalité à celle d'humanité, comme notre définition le fait, n'est pas seulement d'éviter le débat sempiternel de savoir si les machines peuvent penser, avoir des intentions, etc. Car on peut, à tout le moins, supposer qu'une personne peut *déléguer* des décisions ou des intentions à une machine (comme un missile à tête chercheuse *vise bien* sa cible). La question est plutôt de droit. On peut imaginer une situation où du code susceptible d'être compilé* et exécuté avec succès serait généré automatiquement par une machine. Si dans cette action il y a trace d'une intention, que celle-ci soit attribuée à une personne humaine ou à une autre machine, alors on parlerait naturellement d'un programme, mais pas dans le cas contraire – par exemple si cette génération de code survenait suite à une erreur.

Pour Lonati, Brodnik et al. (2022), l'aspect complémentaire à celui d'être une réalisation humaine est d'être un *outil*. Cela veut dire que l'exécution du programme doit servir à des utilisateurs humains – pour travailler, étudier, jouer, etc. Nous retrouvons cette propriété dans la dernière partie de notre définition qui porte sur la visée d'une transformation de la situation. Cependant, comme nous l'avons déjà dit en évoquant l'art informatique, une telle propriété n'est pas strictement nécessaire.

Nous avons également déjà discuté la troisième paire d'aspects – *physicalité* et *abstraction* des programmes – lors de notre commentaire concernant leur nature sémiotique. Cette dualité est inscrite dans la notion même de signes. Cependant, il est intéressant de revenir sur les raisons plus générales pour lesquelles on peut affirmer qu'un programme est une entité abstraite.

Cette proposition peut se référer à de multiples points de la définition. D'abord, un signe peut être lui-même considéré comme une abstraction puisque son essence, comme nous l'avons dit, réside dans son type, qui permet d'identifier les marques concrètes qui lui correspondent. Ensuite, l'*arrangement* des signes peut être considéré comme une propriété abstraite du programme, qui permet de l'inspecter mathématiquement. Enfin, le *comportement* de la machine, on l'a vu, est une notion construite par l'observateur qui abstrait d'un phénomène dynamique

119. (Lonati, Brodnik et al. 2022, p. 11).

les grandeurs pertinentes à considérer. Tout comportement se situe donc dans un espace abstrait des comportements possibles, qu'il est possible de mettre en relation avec les structures possibles de signes. C'est en tous ces sens qu'on peut dire qu'un programme est un objet abstrait.

Cependant, ce n'est pas par là que l'activité de programmer requiert l'abstraction. On peut le voir par un exemple outrageusement simple, où l'on programmerait une machine à reconnaître des images par apprentissage. En supposant que ces tâches de reconnaissance soient extrêmement simples (comme reconnaître des couleurs primaires), et l'algorithme bien rôdé, on peut arguer que le programmeur se contente ici d'alimenter la machine en données, comme le chauffeur alimentait autrefois sa locomotive en charbon. Il n'y a là nulle « manipulation de notions et d'entités abstraites » de la part du programmeur, contrairement à ce qu'affirment Lonati, Brodnik et al. (ibid., p. 11). Que les données aient dû être triées et annotées afin de pouvoir servir à entraîner la machine, cela n'en fait pas des activités abstraites, au moins dans le langage courant qui parlerait tout simplement d'activités de la perception. Si le programmeur fait preuve d'abstraction, dans ce cas extrêmement simple, c'est parce qu'il doit tout de même *réfléchir* à ce qu'il fait. Il peut s'agir, par exemple, de s'assurer qu'il existe suffisamment d'images de chaque objet à reconnaître dans le jeu de données, que la machine ne montre pas de biais, etc. Cet acte de réflexion pourrait être fait *a posteriori*, et par simples tâtonnements d'essais et d'erreurs ; il n'en reste pas moins qu'il y a là une mise à distance de l'acte d'arranger les données, aussi bien que d'observation du comportement de la machine qui peuvent, selon nous, être qualifiée d'abstraction. Comme le dit très bien Blackwell (2002), dans une réflexion sur laquelle nous revenons plus loin :

> [les activités de programmation] ont toutes le caractère élémentaire que l'utilisateur ne manipule pas directement des choses observables, mais spécifie un comportement à produire à un moment futur[120].

Là se trouve, selon nous, le fait fondamental qu'on pourrait appeler *abstraction* dans l'activité de programmer. Cependant, pour notre part, nous préférons l'appeler tout simplement *réflexion*, car il traduit mieux le mouvement même de l'investigation, celui de revenir sur ses propres représentations pour les mettre à distance et en faire des objets susceptibles de critique.

120. (Blackwell 2002, p. 3).

§ 43. Difficultés

Peut-on néanmoins affirmer avoir cerné l' « essence » de la programmation par cette définition ? Si tel est le cas, elle devrait pouvoir rendre compte des de ses transformations possibles, que nous avons regroupées tout à l'heure[121] sous quatre chefs :

1) D'abord, comme nous l'avons déjà remarqué, la définition est neutre quant à la nature de la machine programmée – celle-ci peut donc être aussi bien quantique que moléculaire ou bionique. Elle convient donc à ces formes non conventionnelles de programmation que la recherche explore actuellement.

2) Concernant l'apprentissage machine*, il faut d'abord comprendre que le programme n'est pas ici l'algorithme d'apprentissage, qui est déjà pré-programmé et qui fait partie de la machine. Il réside plutôt dans les données d'apprentissage fournies à cet algorithme afin qu'il puisse résoudre le problème qu'elles décrivent – pour que la machine ait le comportement attendu. Le caractère essentiel de ces données, par exemple des images, est qu'elles aient en effet chacune une *structure* sous-jacente, que la machine va apprendre à reconnaître par comparaison et interpolation progressive. La programmation consiste à choisir les données, à paramétrer l'algorithme, à tester les résultats de sa phase d'apprentissage, jusqu'à ce que le comportement de la machine soit adéquat à la visée du programmeur. On peut donc dire que notre définition permet de décrire l'apprentissage machine* comme de la programmation.

3) Elle convient également à la programmation visuelle, au scripting* et au développement ou la modification d'applications sans écriture de code[122], car un arrangement de signes n'est pas nécessairement contraint par la forme textuelle que prennent les langages de programmation classiques.

4) Enfin, les programmes écrits dans des langages spécialisés comme Catala[123], qui semblent souvent faire abstraction de la machine et ne considérer que la résolution pure d'un problème, sont toujours cependant bien écrits *dans l'intention* qu'une machine les interprète et par là rende *effective* cette résolution.

Cependant ces démonstrations semblent un peu forcées et peuvent à bon droit laisser insatisfait, notamment les trois premières. Notre définition, interprétée dans ces directions, semble ouvrir trop grand la porte à ce qu'est un programme et

121. Voir plus haut, § 36.
122. Voir plus haut, § 38.
123. Voir plus haut, § 37.

menacer d'englober toutes sortes d'usages de l'ordinateur et la manipulation de toutes sortes de machines.

Le second risque est sans doute moins grave qu'il n'y paraît. Si la première explication laisse certes dans un flou total ce qu'il faut entendre par *machine* – une question que nous abordons au chapitre 11 – il est cependant vrai que le terme *programme* a toujours été *aussi* appliqué aux machines les plus simples, et ce bien avant l'ère informatique – il suffit ici de penser à la « programmation » de nos appareils électroménagers. En embrassant très large, notre définition pourrait présenter l'avantage de montrer l'unité sémantique profonde des notions de *machine* et de *programme*, et la continuité de l'informatique avec les pratiques de l'industie moderne. Nous revenons sur cette connexion historique au chapitre 5.

La situation est beaucoup plus troublante concernant les deux explications restantes (2 et 3), dont l'argument est similaire : ce qui, *d'un certain point de vue*, ressemble à l'*usage* d'un logiciel (un algorithme d'apprentissage, une application permettant de planifier des comportements complexes de la machine), *d'un autre point de vue* est cependant bien un « arrangement de signes » et donc convient à notre définition d'une *programmation*.

Admettre qu'une interaction avec la machine peut être interprétée aussi bien dans un sens que dans l'autre remet en cause, par une sorte d'effet domino, une bonne part des distinctions que nous avions relevé plus haut, comme celle entre programme et données, entre programme et machine, entre temps de la programmation et temps de l'exécution, qui forment ensemble l'armature sous-jacente de la définition, si bien que celle-ci semble se vider progressivement de son sens.

On peut le voir en prenant l'exemple d'un document écrit en LaTeX, un traitement de texte, (comme ce travail). Le document source est parsemé d'instructions (par exemple pour les italiques, les têtes de chapitre et de sections, les équations). L'utilisateur peut en définir de nouvelles, voire les regrouper en une librairie* qui sera réutilisable. Le texte cible est généré par une compilation*, comme on le ferait pour le fichier exécutable d'un programme classique[124]. Doit-on donc conclure bravement, comme le suggère Mélès, que toute l'écriture de cette thèse était un travail de programmation ? Cela semble trop contraire à l'usage commun.

Mélès met cependant en avant le point essentiel : la distinction entre programme et données est relative, tout comme celle entre programmeur et utilisateur, et toutes les autres que nous avons citées ci-dessus. Il cite à ce propos avec

124. Voir à ce sujet le développement plus détaillé de (Mélès 2022, p. 9-11).

raison l'article célèbre de Peter Moor, qui avait déjà identifié cette relativité dans le couple *matériel / logiciel* :

> Pour une personne et un système informatique donnés, les logiciels seront les programmes qui peuvent être exécutés sur le système informatique et qui contiennent des instructions que la personne peut modifier, et le matériel sera la partie du système informatique qui n'est pas un logiciel. […] Pour l'utilisateur moyen qui programme dans un langage d'application, tel que Fortran, Basic ou Algol, les programmes en langage machine deviennent du matériel. Pour la personne qui exécute un programme d'application, une partie encore plus importante de l'ordinateur est du matériel[125].

Mélès remarque qu'une raison essentielle de cette relativité est un fait technique que notre définition ne mentionne pas, mais qui est pourtant essentiel à la compréhension de l'informatique, à savoir que le programme, dans l'architecture actuelle de l'ordinateur numérique, dite de Von Neumann, est inscrit *sur le même support* que les données de son exécution. Ce fait apparemment anodin et simplement pratique donne toute sa flexibilité à l'ordinateur, en permettant à tout programme de faire appel à des programmes complémentaires qui sont simplement chargés en mémoire comme le seraient de simples données, ou à enregistrer des données qui deviendront à leur tour des programmes. Un simple bit*, `0` ou `1` au cours d'une exécution, fourni par l'utilisateur ou calculé précédemment par le programme, peut ainsi aiguiller l'ordinateur à exécuter deux programmes complètement différents, et pour ainsi dire permettre au programme de s'adapter automatiquement au cours de l'exécution. De manière plus importante encore, ce mécanisme permet d'implémenter la notion mathématique de récursivité, c'est-à-dire d'une fonction qui s'appelle elle-même pour calculer son résultat jusqu'à régresser à une valeur élémentaire donnée, grâce à la capacité qu'il donne à la machine d'insérer à nouveau le code de la fonction dans une pile* (*stack*) d'instructions restant à exécuter.

La notion d'*ordinateur à programme enregistré* (*stored-program computer*) est si importante pour Donald Knuth qu'il alla jusqu'à déclarer qu'avant elle, il n'y avait pas de notion claire de ce qu'était un programme[126].

Les distinctions entre donnée et programme, entre utilisateur et programmeur, et partant toutes les autres, ont donc un caractère « dialectique » : tout programme peut être vu comme une donnée pour un certain programmeur, qui peut être lui-

125. (Moor 1978, p. 215).

126. Cité par (Swade 2011, p. 145). Pour une mise en contexte historique de ce concept, voir (Haigh, Priestley et Rope 2014).

même perçu comme un utilisateur par un autre programmeur, et ainsi de suite. Il faut bien comprendre qu'il n'y a pas vraiment de terme à cette régression – comme si le concepteur des composants électroniques élémentaires était le « Programmeur Maître » parce qu'au fondement de la pratique de tous les autres. Lui-même programme ses circuits en utilisant des logiciels écrits par d'autres programmeurs. L'industrie informatique fonctionne réellement en cercle ou en réseau, chaque partie produisant des outils pour toutes les autres.

Que conclure de ces analyses ? D'abord, doit-on limiter notre définition de la programmation aux machines « à programme enregistré » ? Il ne le semble pas puisque, comme nous l'avons dit, on utilise également le terme pour des machines simples. Sans doute la programmation *intéressante* est-elle limitée à ces machines complexes, mais cette hypothèse ne justifie pas d'y restreindre notre enquête à ce stade.

C'est en tous cas pour elles que cette distinction entre programmation et utilisation apparaît vraiment. Pour une machine simple, l'usage se confond avec l'exécution du programme de la machine : il n'y a pas vraiment d'interaction possible entre elle et l' « utilisateur ». On peut donc arguer que ce n'est pas la notion de programmation qui est remise en cause ici, mais bien celle d'« usage simple » d'un ordinateur. L'usage est construit justement par le programmeur qui vise à rendre l'interaction avec la machine fluide et transparente, à cacher la *machinerie* derrière l'illusion de l'*ustensilité*, comme le montrent les icônes de nos écrans – dossiers, corbeilles, enveloppes, réveils-matins, etc. Les usages ne seraient ainsi que des programmations hautement stylisées, simplifiées et automatisées progressivement par des milliers de programmeurs et d'ergonomes se succédant dans cet effort sans relâche tout au long des quelques soixante-dix années d'histoire des ordinateurs – et cela jusqu'à aujourd'hui, où elles se trouvent finalement condensées dans des gestes élémentaires, comme un clic ou un balayage d'écran, embarquant chacun des milliers de lignes de codes et des dizaines d'abstractions intermédiaires, complètement invisibles à l'« usager »[127].

Une telle vision est certainement possible, et pourrait peut-être être étayée historiquement, si on parvenait à montrer comment la figure de l'utilisateur fut distinguée progressivement de celle de programmeur avec laquelle elle était initialement confondue, et fut d'ailleurs en partie consciemment construite afin de répondre

127. Pour un aperçu de ces « cathédrales » invisibles d'abstractions, voir la « visite guidée » proposée par (Lee 2017, ch. 3-5) des technologies sous-jacentes à une simple page Wikipedia.

aux impératifs économiques de l'industrie. Ainsi s'expliquerait également cette interpénétration profonde entre programmations simples et usages avancés de l'ordinateur, que nous avons essayé de rendre sensible[128].

Cependant, même s'il y a continuité entre programmation et usages de l'ordinateur, il reste qu'on peut tout de même chercher à caractériser les *paramètres* de cette transformation graduelle. L'un d'entre eux nous est déjà apparu. Un programme est en général conçu comme un arrangement d'inscriptions durables plutôt que de signaux volatiles, car on s'attend à ce qu'il puisse être inspecté et réutilisé. Au contraire, l'usage est vu comme une interaction avec la machine, donc est plus naturellement à une communication, ou à un échange de signaux.

Cependant il ne s'agit pas de propriétés nécessaires. Beaucoup d'usages consistent à créer des documents, donc des inscriptions. Réciproquement, certains langages permettent, voire favorisent, la programmation interactive. Ainsi dans Smalltalk l'environnement de développement et d'exécution sont confondus, et programmer consiste à créer des objets et à leur envoyer des messages ; il s'agit d'actions en temps réel, qui sont immédiatement exécutées. On peut bien sûr définir des blocs de code qui ne sont exécutés que globalement plutôt que ligne à ligne, mais le paradigme initial est bien celui d'un contrôle en temps réel de la machine par des signaux qui lui sont successivement envoyés, comme c'est le cas dans l'administration système*.

Ce qui différencie cette forme de programmation d'une simple utilisation est que les actions du programmeur modifient l'état du système de manière à influencer de manière décisive son comportement futur. En d'autres termes, les signaux qu'il lui envoie génèrent des inscriptions *au sein* du système. Là encore il faut préciser de quoi on parle. On pourrait en effet objecter que, même au sein d'un jeu vidéo, les gestes du joueur modifient l'état du système et influent sur son comportement – ici représenté par le mouvement de petites figurines à l'écran. Cependant cet exemple met bien en évidence la différence importante qu'il y a avec la programmation : les gestes du joueur n'influent le comportement futur de la machine qu'au sein du périmètre restreint de la partie en cours. Cette restriction se mesure à la fois en termes de durée – une fois la partie terminée, tous les gestes du joueur sont oubliés – et en termes d'amplitude des comportements exigibles de la machine : ceux-ci sont limités, dans le cas d'un jeu, à ceux prévus à l'avance par son programmeur. Dans le cas d'un traitement de texte, un document a certes un ho-

128. Voir plus haut, § 38.

rizon temporel indéfini – il peut toujours être exécuté à nouveau – mais la gamme de comportements informatiques qu'il permet de commander reste centrée autour de son affichage et de son impression – une programmation pauvre.

La caractérisation de la programmation relativement à l'usage se situerait donc dans l'accès qu'elle aurait à la plus grande gamme des comportements possibles de la machine, y compris quant à l'ouverture de leur horizon temporel futur. Un tel critère est relatif, mais là encore des précisions sont nécessaires. La plupart des machines aujourd'hui sont bridées, c'est-à-dire que le programmeur a un accès restreint à certaines ressources réservées au système d'exploitation, voire au fabricant même de la machine. Quant à exiger la complétude de Turing, c'est-à-dire la capacité qu'aurait le programmeur de simuler une machine de Turing, ce critère souvent invoqué est trop restrictif, puisqu'on parle bien de programmation dans le cas où la machine est un simple automate fini*.

Il ne nous semble pas qu'on puisse trouver un critère qui permettrait de délimiter strictement la programmation de l'usage sur ce gradient de la richesse des comportements possibles et de l'ouverture de leur horizon temporel. Il ne s'agit pas d'un critère logique, ni matériel, mais d'un critère épistémique, qui mesure la complexité de l'interaction avec la machine.

En effet, s'il existe bien un tel mouvement de simplification de la programmation en usages, c'est qu'il doit exister également un mouvement inverse, qu'on pourrait appeler de *remontée vers l'origine*, lorsque par exemple un utilisateur entreprend de déconstruire ses propres usages numériques et de comprendre, voire modifier, les programmes qui les sous-tendent. « Tout ne se vaut pas », donc, et il existe bien des postures de programmation plus complexes que d'autres, qui ont *plus* de contrôle sur la machine, peuvent faire *plus* de choses quant à son comportement, peuvent arranger les signes de *plus* de manières. Quitte à tout décrire comme programmation, notre questionnement revient à se demander quels sont les critères qui nous permettent de juger de ce *plus ou moins* de programmation dans l'interaction avec l'ordinateur.

§ 44. La programmation comme procès épistémique

Dans la recherche de ce gradient de programmation, il est utile de nous appuyer sur la réflexion de Blackwell (2002), un psychologue des interactions homme - machine déjà rapidement évoqué.

Blackwell voit bien le caractère relatif de toute distinction entre usage et programmation d'un ordinateur. Considérant, entre autres, les tableurs*, les traitements de texte, les langages de scripting*, le langage HTML, il demande :

> Lequel de [ces] éléments peut-il sans ambiguïté être catégorisé comme langage de programmation ou comme une autre forme de logiciel ? […] Les distinctions entre ces technologies ne sont pas du tout nettes. Toutes sont capables de produire des documents texte modifiés dynamiquement, et plusieurs peuvent potentiellement être utilisées pour créer des résultats apparemment identiques. Pourtant, certaines sont classées dans les contextes professionnels comme étant des langages de programmation, et d'autres non[129].

Cependant, pour lui, la difficulté d'opérer ces distinctions provient du fait que ce ne sont pas les technologies utilisées qui caractérisent la programmation, mais l'*expérience* de l'utilisateur, qui doit réaliser des tâches plus ou moins complexes cognitivement pour parvenir à ses fins – comme spécifier ce qu'il veut obtenir comme résultat, caractériser la situation favorable pour cela, identifier des exceptions, comprendre les règles du fonctionnement élémentaire de la machine qu'il s'agit de modifier, etc. Selon que ces tâches sont ardues ou non, il y a lieu de parler de programmation, et sinon simplement d'utilisation :

> La proposition de cet article est que lorsque les gens disent qu'ils programment, nous ne devons pas nous demander si cette activité est de la programmation véritable, mais plutôt analyser leur expérience afin de comprendre la nature générale de l'activité de programmation[130].

Cette idée est capitale. Elle nous invite à ne pas centrer notre interrogation sur les outils et les productions du programmeur, mais sur son procès intellectuel lui-même. Elle suggère que, si nous nous sommes dotés d'une définition littéralement correcte de la programmation, celle-ci n'explique pas cependant le procès épistémique qui seul la caractérise véritablement, et qui permettrait de la distinguer d'un simple usage ou de l'entraînement par apprentissage d'un algorithme.

Pour Blackwell, la première caractéristique de la programmation relativement à l'usage simple de l'ordinateur est la perte de la manipulation *directe* des éléments de la situation. Le programmeur doit prévoir *ce qui arrivera plus tard* à la situation, et cela peut impliquer qu'il examine de nombreuses situations différentes, dont la possibilité n'est pas immédiatement visible. Ce caractère *indirect* du contrôle du comportement de la machine implique l'usage de notations pour en décrire les

129. (Blackwell 2002, p. 2).
130. (ibid., p. 3).

règles. Elles permettent de créer des renvois indirects à plusieurs niveaux, comme on dit d'un coup de billard qu'il est « à plusieurs bandes ». Blackwell contraste l'exemple de la programmation d'un vidéo-enregistreur qui requiert de simplement spécifier l'heure et le canal (programmation simple, proche d'un simple usage) de celle d'une chaudière, dont on peut définir plusieurs modes opératoires, puis spécifier le choix du mode en fonction des jours de la semaine. On a créé ici une abstraction (le mode) que l'utilisateur peut à présent manipuler comme si elle était un objet, lui permettant de ne pas répéter les mêmes instructions pour chaque journée[131].

Ce qui distingue donc essentiellement la programmation informatique de programmations plus simples est cette flexibilité indéfinie de créer du contrôle *indirect* de l'action, à autant de niveaux que souhaitable. Blackwell appelle cela *abstraction*, terme auquel nous préférons celui de *réflexion*, moins ambigu, car le travail spécifique du programmeur n'est pas, par exemple, de trouver des points communs à une collection d'objets, mais de se mettre à une certaine distance d'un processus afin de formuler les conditions de la diversité de ses comportements – conditions qui peuvent elles-mêmes résulter de processus, et ainsi de suite.

Le programme, tel que le présente notre définition, n'est que le résultat de ce travail de la réflexion. On peut toujours appeler *programme* toute manipulation simple d'une machine à l'aide de notations consignées par avance : c'est à ce titre que peuvent être ainsi désignés, par exemple, la minuterie d'un appareil électroménager ou une recette de cuisine. Cependant ne doit nous intéresser, comme mouvement propre de la *connaissance* informatique, que cette possibilité de la réflexion pratique humaine de s'organiser en procédures enchâssées logiquement, qui semble n'être apparue que récemment[132] et qui reste à ce stade mystérieuse. Ce sont donc seulement des programmes *complexes* qui doivent retenir notre attention.

À ce titre, il nous faut revenir aux programmes écrits dans des langages spécialisés, et à notre explication 4) donnée au début du développement précédent. Celle-ci est la seule qui soit vraiment convaincante, car il existe bien une tension essentielle entre les deux visées présentes en tout programme, celle qui porte sur le comportement de la machine et celle qui s'intéresse au problème à résoudre. Selon le point de vue qui est adopté, l'une ou l'autre est mise en avant, selon qu'on

131. (ibid., p. 4).
132. Voir plus haut, § 13.

vise à expliciter les stratégies de résolution de problème ou à optimiser l'efficacité du comportement de la machine. C'est cependant leur contiguïté qui pose mystère, et que nous ne pouvons seulement expliquer, comme le faisaient les premiers informaticiens, par la simple juxtaposition de la planification et du codage, de l'algorithme et de la routine. Il nous faut entrer plus avant dans ce processus même de « traduction » d'un domaine de significations à l'autre, expliquer comment, selon la belle définition de la programmation par Alexandron et al. (2014), il est possible de « prendre un problème défini dans son domaine propre et à construire une solution en utilisant les outils du domaine de la solution [*i.e.* de la machine][133] ». C'est à l'étude de cette transmutation des significations qui se fait au cours du processus de la programmation qu'est consacré le chapitre suivant.

133. (Alexandron et al. 2014, p. 21:2).

Chapitre 3

Raisonner avec les machines

§ 45. Introduction

Le chapitre précédent nous a donné l'idée de la difficulté à cerner la nature de la programmation tant cette activité prend des formes différentes. Cela se voit dans la difficulté à cerner l'*objet* même de cette activité. Un programmeur à qui on poserait la question *« Que fais-tu ? »* aurait le choix au moins entre les réponses suivantes :

1. *« J'écris un programme en* Python *(ou un autre langage) »*.
2. *« Je programme un ordinateur »*.
3. *« Je résous un problème »*.

On peut reconnaître là les trois moments de la définition que nous avons donnée au chapitre précédent, selon que la réponse met en valeur l'acte d'arrangement des signes (*« j'écris... »*), sa finalité immédiate de réguler le comportement d'un ordinateur, ou encore sa finalité prochaine qui est de résoudre un problème dans une situation donnée.

Nous avons vu – c'est le point de départ de notre interrogation – que les problèmes auxquels la programmation pouvait apporter son assistance étaient d'une très grande variété, et appartenir aux domaines de l'ingénierie, de la finance, des sciences, etc. Cependant, une classe particulière de problèmes a toujours été particulièrement associée à l'informatique, celle des problèmes mathématiques. Dans ce domaine, il est possible d'argumenter que les réponses 1 et 3 convergent, autrement dit que l'écriture de programmes et la résolution de problèmes formels sont de nature similaire. Trouver une preuve et un algorithme, puis les formaliser dans un langage précis, ne sont pas des activités très différentes.

C'est dans ce cadre qu'a été élaborée la conception la plus répandue de la programmation. Programmer, ce serait ainsi résoudre des problèmes formels, dans un langage symbolique contraignant, qui a l'avantage de pouvoir automatiser le calcul de leurs solutions par un ordinateur.

Dans cette interprétation, la réponse 2 est perçue comme assez superficielle. L'ordinateur a un rôle de simple exécutant dans cette affaire, et s'il présente certaines contraintes ou risques d'erreurs, celles-ci sont en général négligeables. La programmation est justement cette technique dont l'objet est presque entièrement défini par ses propriétés fonctionnelles, dit Herbert Simon, de telle sorte que ses propriétés matérielles sont presque toujours « hors sujet »[1]. Un aphorisme célèbre, faussement attribué à Edsger Dijkstra, un théoricien de l'informatique, affirme :

> L'informatique ne traite pas plus d'ordinateurs que l'astronomie ne traite de télescopes[2].

Quant aux problèmes du second type évoqué, ceux des ingénieurs, des financiers, etc. selon cette interprétation, ils ne peuvent recevoir une solution informatique que lorsqu'ils ont été proprement formalisés ou au moins reformulés dans un langage intermédiaire. Ils se présentent alors sous forme de *spécifications*, qui sont seules pertinentes pour le programmeur.

Ce chapitre remet en cause cette conception, en argumentant que les problèmes formels sont seulement une classe très spécifique des problèmes que savent résoudre les ordinateurs et que la « barrière » du langage symbolique, la seule que saurait comprendre un ordinateur, n'en est pas vraiment une. Au contraire, la programmation est toujours une programmation *de machine*, et cette machine est elle-même un élément dans une situation réelle au sein de laquelle se pose le problème à résoudre, et avec laquelle elle interagit.

Nous commençons par exposer l'interprétation de la programmation comme activité formelle, en nous intéressant au développement des méthodes formelles, ainsi qu'à leurs critiques (section 3.1). Nous présentons ensuite (section 3.2) trois exemples de projets de programmation qui s'accordent mal avec cette interprétation. Dans la section suivante 3.3 nous présentons et illustrons un cadre de réflexion proposé par l'informaticien Michael Jackson qui explicite les relations entre expression de besoins, spécifications et programme. Il nous permet de développer, à la section suivante 3.4, une interprétation nouvelle qui, tout en don-

1. (Simon [1991] 1996b, p. 17).
2. Pour l'origine de cet aphorisme, voir ce site.

nant toute leur place aux méthodes formelles, présente d'abord la programmation comme la négociation et la préparation d'une interaction entre deux états de choses, la situation où se pose le problème et la machine où se programme une solution. Nous soulevons une série d'objections à cette interprétation, dont une partie ne pourra être résolue que par des développements ultérieurs de notre réflexion.

Il est utile, avant de commencer, de préciser en quel sens nous utilisons le terme *formel* dans nos développements, en nous référant à l'étude de Dutilh Novaes (2011) qui documente ses différents emplois en philosophie. Notre usage est circonscrit à la conjonction de deux d'entre eux, d'ailleurs étroitement associés : d'abord, celui de *formel* comme « dé-sémantification », ou « abstraction de la signification » : la forme est ici celle des signes eux-mêmes. Ce sens est tardif dans la longue histoire du terme et se développe principalement avec le programme formaliste de Hilbert. C'est de lui que découle le second sens qui nous intéresse, celui du *formel* comme *calculable*, c'est-à-dire régi par des règles entièrement explicites. Dutilh Novaes a raison de distinguer les deux sens, même s'ils sont historiquement associés. Elle classe le premier parmi les sens de *formel* qui s'opposent à *matériel*, et le second parmi ceux qui s'opposent à *informel*. Les premiers ont trait à la notion de forme et relèvent de la logique des entités, les seconds on trait à la notion de *règle* et relèvent plutôt de la logique des actions et des normes. Aussi a-t-elle également raison de souligner qu'il est possible de décrire des systèmes de calcul opérant sur des objets signifiants – donc non formels dans le premier sens. C'est ce qui rend possible la critique d'une approche formelle de la programmation, par laquelle nous commençons ce chapitre.

3.1 L'interprétation formelle de la programmation

Dans cette section, nous présentons une interprétation assez répandue de la nature de la programmation, ainsi que ses faiblesses. Un cadre utile pour cette discussion est la définition assez récente du concept de programme par le philosophe et théoricien de l'informatique, Ray Turner, comme d'un « paquet ontologique » constitué d'une spécification, d'un texte symbolique, et d'une implémentation pour un système informatique donné (§ 46). Nous montrons ensuite comment ce cadre permet, au prix de certaines précisions, d'interpréter la programmation comme une activité intellectuelle formelle encadrée par des « pare-feu logiques » qui la

rendent indépendante de contingences matérielles de deux types : d'une part les intentions subjectives qui président à la détermination de l'objet du programme, d'autre part le bon état de marche de l'ordinateur qui exécute celui-ci. Nous lions cette interprétation à l'émergence historique d'une certaine approche de la programmation, les méthodes formelles* (§ 47). Nous passons ensuite en revue les critiques de cette approche qui, bien qu'elles se trouvent réfutées *de facto* par le succès des méthodes formelles, pointent selon nous vers l'inadéquation de l'interprétation de la programmation à laquelle elles sont associées (§ 48 et § 49).

§ 46. Le programme, un paquet de trois parties

Dans son ouvrage d'introduction à la philosophie de l'informatique, Ray Turner présente un programme comme l'association d'une spécification, d'un texte symbolique (le programme écrit en langage de programmation de haut niveau) et d'une implémentation pour un système informatique donné. De ce fait, la programmation se présente comme une activité constituée essentiellement de deux étapes, permettant de transformer ces éléments l'un dans l'autre. Il la représente donc dans un schéma similaire à celui de la figure 3.1[3]. Dans ce schéma, la spécification décrit le problème à résoudre dans toutes ses composantes, comme l'objectif à atteindre, les moyens disponibles et les contraintes à respecter. Le programme symbolique est un texte qui, correctement interprété, présente une procédure de résolution du problème. La configuration est l'état physique de la machine qui permet l'exécution de la procédure : par exemple, le programme, compilé* en langage-machine*, est présent dans sa mémoire vive*, et les ressources nécessaires à son exécution lui sont accessibles. Ces trois objets sont donc bien physiques, la spécification et le programme étant des textes qui existent sur papier ou sous forme numérique.

Les deux étapes de la conception et de l'implémentation correspondent aux travaux du programmeur. La conception peut elle-même être découpée en deux activités, la résolution du problème *et* sa rédaction dans un certain langage symbolique. La première activité est algorithmique (il s'agit de trouver ou de choisir et d'adapter un algorithme au problème en jeu) et la seconde est appelée communément *codage*. L'implémentation est la préparation de l'ordinateur pour l'exécution du programme, ce qui n'implique pour les programmes simples que sa compilation*,

3. (Turner 2018, p. 52).

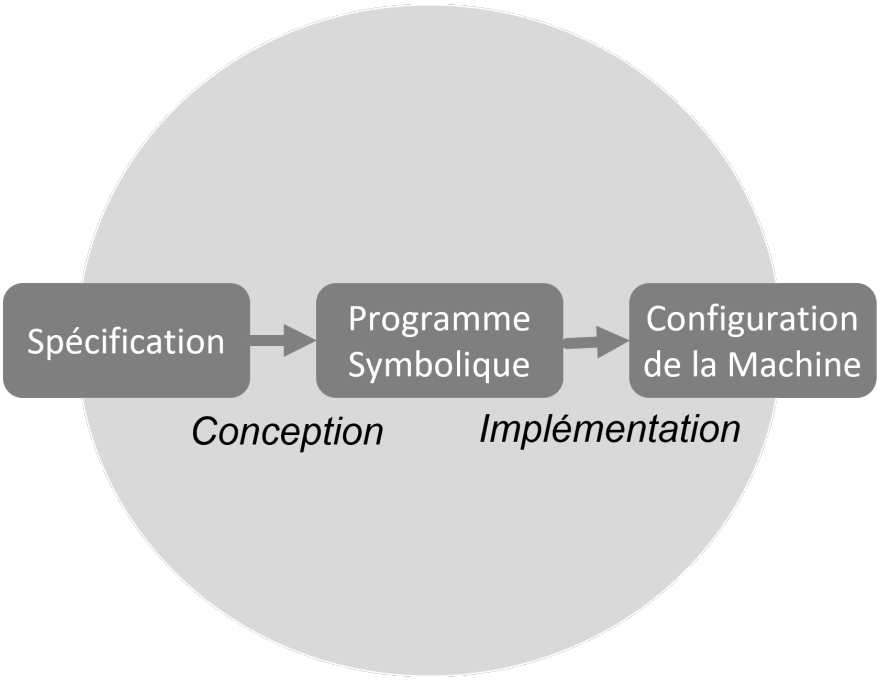


FIGURE 3.1 – La structure tripartite d'un programme selon Ray Turner

mais qui peut comprendre de nombreuses autres tâches, comme la préparation de bases de données, de périphériques, d'interfaces avec d'autres systèmes, etc.

D'où vient une pareille structure ? Turner, en employant le terme *artefact* pour désigner le programme-machine ou le processus de son exécution, suggère qu'elle reflète la structure générale de l'ingénierie de production. Dans cette discipline, la spécification traduit la finalité fonctionnelle de l'objet à inventer. La conception consiste à déterminer une structure physique (mécanique, chimique, etc.) répondant à cette spécification. L'implémentation consiste à construire l'artefact concret. Ainsi, selon Turner, le programme symbolique correspond à une forme de structure. Turner déclare explicitement s'inspirer ici de l'épistémologie de la technique du philosophe Peter Kroes, qui présente un tel schéma conceptuel[4].

L'idée de prendre l'ingénierie pour modèle de la programmation remonte, on l'a vu[5], à la « crise du logiciel » et aux conférences de l'OTAN de la fin des années 1960. Il fallait professionnaliser la programmation pour augmenter la fiabilité des logiciels, et pour ce faire le caractère méthodique et relativement standardisé des

4. Voir par exemple (KROES 2012, p. 127-129).
5. Voir plus haut, § 32.

méthodes d'ingénierie apparaissait exemplaire. En particulier, le métier d'ingénieur est circonscrit par des interfaces clairement identifiées, qui ont inspiré la méthode de la modèle en cascade[*6]. D'un côté, il est en dialogue avec un client lui exprimant ses besoins, et de l'autre avec des équipes de production exécutant ses instructions. Au sein même de ce métier, on peut distinguer la phase de la conception et la phase d'industrialisation d'un produit, ce qui correspond donc bien au schéma de Turner.

Ce schéma a cependant une autre source, qui conçoit la programmation comme une activité mathématique, devant être dirigée par des méthodes formelles[*]. L'idée, ici, est qu'en établissant la spécification du programme et celle de la machine comme *pare-feu logiques* entre le programme et la réalité extérieure, ce dernier allait pouvoir acquérir un degré de fiabilité similaire aux démonstrations du mathématicien.

§ 47. Les pare-feu logiques de Dijkstra et Hoare

Nous empruntons l'expression de « pare-feu logique » au théoricien de l'informatique Edsger Dijkstra, qui milita ardemment pour une conception mathématique de la programmation.

> Le rôle d'une spécification fonctionnelle formelle est simplement d'agir comme un pare-feu logique entre deux préoccupations complètement différentes, connues sous les noms de « problème de l'agrément » et « problème de la correction ».
> Le problème de l'agrément[7] consiste à savoir si un système qui répond à une spécification fonctionnelle formelle donnée satisfait nos besoins, répond à nos attentes ou réalise nos espoirs. Le problème de la correction consiste à savoir si la conception [d'un programme] répond à une spécification fonctionnelle formelle donnée[8].

Le programmeur, pour Dijkstra, a uniquement pour compétence et mission de résoudre des problèmes de correction, c'est-à-dire de produire des programmes qui répondent exactement à une spécification. Le problème de l'agrément fait appel implicitement à une autre figure, celle du maître d'ouvrage du projet, qui porte ces attentes et ces espoirs auxquels fait référence Dijkstra. Le programmeur et le maître d'ouvrage désignent ici des rôles logiques, ce qui n'implique pas qu'il s'agisse de deux personnes distinctes (quelqu'un qui se donne à lui-même une tâche

6. Voir plus haut, § 32.
7. *Pleasantness problem*. L'ambiguïté du terme français *agrément* a l'avantage de signifier aussi bien la note ironique voulue par Dijkstra que le terme plus précis de *validation* auquel il se réfère.
8. (Dijkstra 1986).

de programmation est tour à tour maître d'ouvrage et programmeur), ni même qu'il s'agisse de personnes tout court (il peut s'agir d'équipes ou d'organisations, voire de machines), et encore moins qu'il s'agisse d'une relation commerciale ou pécuniaire.

Une note terminologique est ici utile. Le *maître d'ouvrage* désigne le donneur d'ordre et le bénéficiaire d'un service. C'est lui qui exprime le besoin, choisit le prestataire de service et accepte sa réalisation. Il s'agit d'un terme plus exact que celui de *client*, qui garde une connotation commerciale. *Utilisateur* serait également impropre, car par exemple, dans le cas du système d'information d'une organisation, les utilisateurs sont le plus souvent des employés qui ne sont pas en position de décision ; ils peuvent au mieux exprimer leurs besoins spécifiques et leur point de vue. Le maître d'ouvrage est dans ce cas de figure la direction de l'entreprise, qui délègue en général la maîtrise d'ouvrage à un comité de pilotage du projet. Enfin, dans le cas d'un logiciel publié par un éditeur, comme par exemple Microsoft Word, ni le client ni l'utilisateur n'ont participé à l'élaboration du logiciel ; il ne peuvent que décider de l'acheter ou non. Le maître d'ouvrage est ici la direction générale de Microsoft, ou encore le responsable de la ligne de produit Word. Dans la suite de nos réflexions, nous utiliserons donc systématiquement de *maître d'ouvrage* et *programmeur* pour désigner les deux rôles complémentaires d'un projet de programmation.

Il peut y avoir un écart entre la correction du programme et l'agrément du maître d'ouvrage parce que les spécifications ont pu mal transcrire le besoin, par exemple en oubliant des détails ou des situations particulières. Le maître d'ouvrage peut aussi avoir changé d'avis en cours de route – peut-être même en inspectant le résultat. Tout cela, pour Dijkstra, n'est pas l'affaire de la « science » de la programmation.

Cette idée est passée dans le vocabulaire de l'ingénierie logicielle sous la forme d'une distinction entre *validation* et *vérification*, comme l'explique par exemple ce manuel concernant l'expression de besoins en informatique :

> La validation et la vérification sont deux des activités les plus critiques lors du développement d'un produit. Ces noms sont attribués aux activités qui s'assurent que ce qui est développé répond réellement aux attentes des clients. […] La validation s'assure que ce qui est développé répond aux besoins du client qui paie pour le produit ; la vérification s'assure que le produit se comporte comme spécifié. Cette différence est clairement expliquée dans [Boeh1979], en utilisant ces deux questions :
> - Construisons-nous la bonne chose ? (Validation)

> - Construisons-nous bien la chose ? (Vérification)[9]

Le « problème de l'agrément » constitue donc la borne externe de l'activité de programmation, si on la considère du côté de la spécification (voir figure 3.1.).

Du côté de l'implémentation, on peut identifier un autre problème qui sort également de la compétence du programmeur selon Dijkstra, celui de la fiabilité du matériel informatique devant exécuter le programme. Qu'un fusible saute, que deux périphériques se révèlent être incompatibles entre eux, qu'il y ait une mauvaise conception du processeur, toutes ces choses peuvent faire échouer le programme mais n'enlèvent rien à sa correction. Turing, qui fit cette remarque très tôt, proposa ainsi de distinguer entre *erreurs de conclusion* et *erreurs de fonctionnement*, suggérant que les programmeurs avaient essentiellement affaire à des *machines abstraites* :

> [Il y a] une confusion entre deux types d'erreur. On peut les appeler « erreurs de fonctionnement » et « erreurs de conclusion ». Les erreurs de fonctionnement sont dues à une défaillance mécanique ou électrique qui fait que la machine se comporte différemment de ce pour quoi elle a été conçue. Dans les discussions philosophiques, on aime ignorer (*one likes to ignore*) la possibilité de telles erreurs ; on discute donc de « machines abstraites ». Ces machines abstraites sont des fictions mathématiques plutôt que des objets physiques. Par définition, elles sont incapables d'erreurs de fonctionnement[10].

En d'autres termes, un programmeur pourrait effectuer des erreurs de conclusion, mais il ne serait pas pertinent de l'accuser d'erreurs de fonctionnement de la machine : de telles erreurs seraient plutôt le fait du fabricant de l'ordinateur ou du responsable des opérations informatiques.

En synthèse, il se dégage de ces réflexions l'idée d'une division du travail entre trois rôles, où le programmeur occupe la place centrale, entre un maître d'ouvrage qui établit un cahier des charges (les spécifications fonctionnelles) et un fabricant informatique qui livre une machine conforme à des spécifications matérielles. Le rôle du programmeur peut alors être précisé : il s'agit de construire une transformation des spécifications fonctionnelles *dans les termes* des spécifications matérielles, ce schéma pouvant être *prouvé* correct.

Avant de passer à la discussion et à la critique d'une telle conception de la programmation, il est utile d'en préciser la genèse historique.

9. *Are we building the right thing ? (Validation) // Are we building the thing right ? (Verification).* (Hood et al. 2008, p. 157).
10. (Turing 1950, p. 449).

Comme le rappelle Mark Priestley dans son histoire de l'informatique[11], la correction des programmes désignait d'abord l'adéquation globale du résultat, c'est-à-dire du processus informatique, avec l'enchaînement des opérations déterminé par le programmeur. Avec l'automatisation progressive de la programmation[12] et l'apparition des langages de programmation de haut niveau, une notion nouvelle de la correction apparut, désignant « une forme particulière de cohérence entre un programme et sa spécification[13] ». Cette évolution était dans une certaine mesure dictée par le pragmatisme même : au fur et à mesure que la fiabilité du matériel informatique augmentait, la source majeure des erreurs ne résidait plus dans les défauts électriques et matériels des ordinateurs, mais dans l'activité même de programmation, c'est-à-dire dans la conception d'algorithmes et dans leur codage en séquences d'instruction. Ce genre d'erreurs était jugé insupportable, car il gaspillait inutilement du temps de la machine, qui était rare et cher à l'époque : on avait intérêt, lorsqu'on y avait accès, à présenter un programme correct. Par ailleurs la complexité grandissante des programmes montrait la limitation essentielle d'une méthode empirique de tests, qui ne pouvait plus explorer toute la combinatoire de leurs configurations possibles[14]. La tâche fastidieuse du codage de l'algorithme en instructions machine était une source d'erreurs importante, et elle conduisit progressivement au développement des langages de haut niveau, qui l'automatisaient en partie, en invitant le programmeur à s'exprimer dans un langage plus directement adapté à son problème. Cependant l'importance de la bonne correction de ces outils de codage automatique – qui allaient bientôt s'appeler compilateurs* – grandissait d'autant, puisqu'une erreur en leur sein compromettait tous les programmes les utilisant. Par ailleurs, on s'aperçut que les programmes écrits en langage de haut niveau continuaient également à recéler des erreurs, et qu'il fallait les vérifier en tant que tels avant de les soumettre à la compilation. La première voie suivie fut celle du débogage* (*debugging*), c'est-à-dire de tests automatiques conduits pendant la compilation* pour déceler les erreurs les plus génériques, comme l'incohérence des types* de données. Mais cela ne suffisait pas à déceler les erreurs de conception, et l'idée se développa donc d'élaborer des méthodes *prouvant a priori* (c'est-à-dire avant compilation) l'adéquation d'un programme à ses spécifications, par inspection simple de son texte ou de ses dia-

11. (Priestley 2011, p. 256-73).
12. (ibid., ch. 7-9).
13. (ibid., p. 253).
14. (ibid., p. 256-257).

gramme de flux*.

Ce programme de recherche se déploya d'abord avec optimisme au cours des années 1960, grâce aux méthodes dites *sémantiques* développées par Floyd, Hoare, et Dijkstra. Il s'agissait d'élaborer un traitement de la vérification par des calculs logiques formels, de telle sorte que le programme gagne la même certitude que celle d'une preuve. C'est à partir de là que l'idée d'une spécification formelle des programmes se dégagea, puisqu'elle devait pouvoir être présentée comme un théorème à démontrer et qu'elle se substitua, au moins dans l'esprit des partisans de cette approche, à celle de la finalité externe du programme[15].

Cette interprétation fait donc de la programmation une activité similaire à un raisonnement formel. La phase de tests, c'est-à-dire la confrontation empirique d'un programme à la réalité de son exécution, traditionnellement tenue pour vitale dans les projets de développement, n'est pas ici intrinsèque à l'activité de programmation mais seulement un pis-aller méthodologique. L'objet véritable de la programmation réside donc dans cet espace formel généré par l'écart entre une spécification fonctionnelle et des spécifications matérielles. Elle tient tout entière dans l'idée que, *si* la machine fonctionne conformément à ses spécifications, et *si* la spécification du problème à résoudre correspond aux besoins du maître d'ouvrage, *alors* l'écriture du programme formel doit suffire à satisfaire ce dernier.

Cette interprétation est cohérente avec l'explication que Denning donne de la capacité qu'ont les ordinateurs à transformer l'information, exposée en introduction[16], que nous rappelons ici :

> Le processus de conception [du programme] transfère l'idée d'addition de nos têtes vers des schémas d'instruction qui effectuent l'addition, [et] la signification de l'addition est conservée dans la conception de la machine et de ses algorithmes.

On retrouve dans ces deux phrases l'idée que l'effectivité d'un processus informatique dépend d'une double correction, celle d'un programme à une idée (la spécification formelle), et celle (sous-entendue) d'une machine réelle à sa conception. La programmation est donc la conception d'équivalences entre des opérations mentales (sémantiques) et des opérations mécaniques. Le langage de spécification représente ici le pôle sémantique et le langage de programmation celui de la machine ; programmer, c'est effectuer la traduction d'un langage à un autre, ou encore si on l'exprime mathématiquement, établir un isomorphisme d'une structure formelle à

15. (Priestley 2011, p. 264-265).
16. Voir plus haut, § 7.

une autre.

§ 48. Les critiques des méthodes formelles

Il convient de noter que cette interprétation n'est pas celle de Ray Turner. On le voit à la différence entre le schéma initial 3.1 et le schéma étendu 3.2, qui fait apparaître les phases externes de l'expression de besoins et de l'exécution. Cette ex-

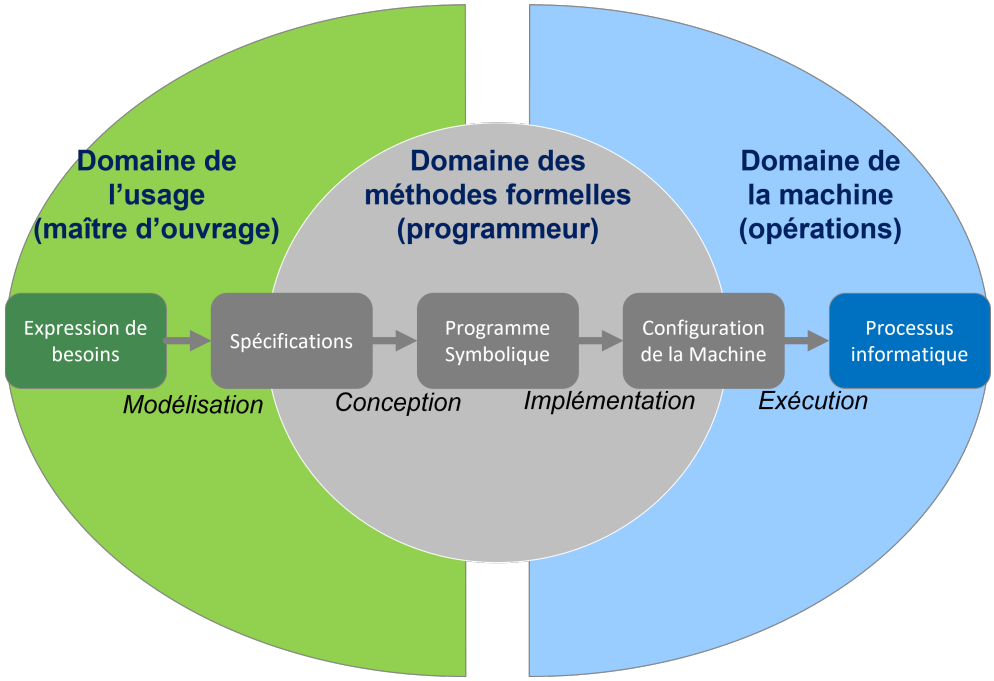


FIGURE 3.2 – Interprétation de la programmation selon les méthodes formelles

tension soulève de nouvelles questions. Du côté de l'implémentation, cela est assez évident, car il semble qu'on n'a fait que repousser le problème. Si le programmeur se contente de fournir un programme répondant aux spécifications d'une machine abstraite, qui vérifie alors que la machine réelle est conforme à cette spécification ? Du côté de la conception du programme, il semble également qu'on a à présent deux problèmes au lieu d'un. En effet, afin que la correction d'un programme puisse être vérifiée formellement relativement à sa spécification, il faut que celle-ci soit également exprimée formellement, et que sa cohérence interne soit aussi vérifiée – car on ne pourrait prouver la cohérence d'un programme avec une spécification contradictoire ! Apparaît donc ici un *second* travail non négligeable de

réflexion situé en amont de la conception du programme, dont on peut se demander s'il est partie de la programmation ou non. Ce schéma plus complexe oblige à distinguer plus précisément entre l'*expression de besoins*, communiquée en général en langage naturel, qui reste le fait du maître d'ouvrage, et la spécification formelle proprement dite. Le passage de l'une à l'autre est appelée *modélisation*[17] ou *analyse*[18] et elle est souvent le fait de personnes ayant des compétences spécialisées, capables de comprendre les besoins du maître d'ouvrage aussi bien que ceux du programmeur.

Ces questions concernant la pertinence de placer de tels « pare-feu logiques » autour de l'activité de programmation furent sous-jacentes à l'important débat critique que suscitèrent les méthodes formelles dans les années 1980, qui s'inscrit dans le débat disciplinaire plus large évoqué plus haut[19]. De nombreux programmeurs ne reconnurent pas la réalité de leur pratique dans cette approche mathématique, qu'ils accusèrent donc d'être déconnectée du réel et dangereuse pour le progrès de leur discipline et sa professionnalisation, qu'ils voulaient rapprocher de l'ingénierie.

Dans un article qui fit date, De Millo, Lipton et Perlis (1979), des informaticiens réputés, attaquèrent la vision de Dijkstra et de Hoare sur la base d'arguments principalement pragmatiques. Les spécifications des programmes qui répondent à des besoins concrets dans le monde réel, notamment dans le monde industriel et économique, diffèrent de deux manières essentielles des problèmes mathématiques. D'abord, chacun des éléments de ces spécifications complexes est en soi très simple à appréhender – il s'agit par exemple d'une règle calculant la prime d'un employé en fonction de son ancienneté. Il y a donc rarement besoin d'une preuve pour vérifier que le programme code bien la spécification[20], contrairement au cas du théorème mathématique où la preuve sert à établir un résultat non évident, souvent grâce à une intuition qui requiert d'être examinée en détail. Ensuite, ces spécifications s'étendent souvent sur des milliers de lignes[21], et c'est donc leur propre cohérence et complétude qui est en jeu. Contrairement au théorème mathématique, dont l'énoncé tient en général en quelques lignes, on n'est jamais sûr de l'objet même de ce qui est à démontrer, ni qu'il est tout simplement

17. (Turner 2018, p. 121).
18. (Constantinidis 2015, p. 105).
19. Voir plus haut, § 12.
20. (De Millo, Lipton et Perlis 1979, p. 276).
21. (ibid., p. 276).

cohérent. Le simple ajout d'une clause minime à une spécification suffit à compromettre sa cohérence logique[22]. L'évolution constante des besoins rend donc toute entreprise de vérification vaine et non économique :

> Un chercheur suffisamment fanatique pourrait être disposé à consacrer deux ou trois ans à la vérification d'un logiciel important s'il pouvait être assuré que le logiciel resterait stable. Mais les programmes de la vie réelle doivent être maintenus et modifiés. Il n'y a aucune raison de croire que la vérification d'un programme modifié sera plus facile que la vérification initiale[23].

B. C. Smith (1985) développe ces arguments en insistant sur un point précis : dans le cas des programmes complexes, la difficulté se situe dans la spécification elle-même – qu'elle modélise bien ce qu'on souhaite réaliser, et à quelles conditions. Bien que les méthodes formelles puissent servir à explorer la cohérence interne et la complétude de la modélisation, elles sont tout de même impuissantes à résoudre son problème central, à savoir que l'expression de besoins initiale ait omis des informations essentielles. Ce fut le cas par exemple lors de la fausse alerte nucléaire de 1960[24], lorsqu'un radar fut perturbé par la luminosité de la Lune. Les meilleures méthodes formelles du monde ne peuvent pallier un tel oubli. Cette difficulté n'est rien d'autre que le problème du cadrage rencontré par l'intelligence artificielle[25], et que Smith illustre par l'évocation de cas limites : jamais le mode d'emploi d'un réfrigérateur ne mentionne qu'il ne fonctionnera pas correctement si on branche un grille-pain à l'intérieur, si on l'installe à l'envers, ou si on l'envoie dans l'espace[26]. Une « myriade d'hypothèses » similaires de sens commun sont pareillement omises dans l'expression de la spécification la plus précise.

L'article de James Fetzer (1988) déchaîna la polémique, peut-être parce qu'il était écrit par un philosophe, qu'on pouvait plus aisément charger d'incompétence footnoteOn peut consulter à ce sujet les lettres adressées aux *Communications of the ACM* (vol. 32, n. 3, 1989, p. 374-381), protestant contre la publication de l'article.. Pourtant, l'argument de complexité, qu'il présente de multiples manières, n'est pas essentiellement différent de ceux de De Millo, Lipton et Perlis (1979) et de B. C. Smith (1985) :

1. D'abord, Fetzer (1988, p. 1055) oppose la complexité cumulative de pro-

22. (ibid., p. 278).
23. (ibid., p. 277-278).
24. (B. C. Smith 1985, p. 18).
25. Voir plus haut, § 9.
26. (B. C. Smith 1985, p. 23).

grammes structurés tels que Hoare et Dijkstra les conçoivent – des emboîtements bien ordonnés de programmes plus simples – à la complexité « en patchwork » qu'on trouve dans les programmes « de la vie réelle » tels qu'ils sont décrits par De Millo, Lipton et Perlis dans la citation donnée plus haut. Ce type de complexité ne permettrait pas à l'entreprise de vérification d'être additive – de se baser sur la vérification de ses parties.

2. Plus loin il avance que l'exécution du programme le plus simple est dépendante d'un environnement causal complexe – « le compilateur, le processeur, l'imprimante, le papier, et tout autre composant dont les propriétés spécifiques influencent l'exécution du programme[27] ». Ici la complexité est celle de l'environnement matériel dont les bornes sont indéfinies – et dépassent le système informatique proprement dit (« le papier »).
3. Dans une dernière mention de l'argument de complexité, il évoque les systèmes avioniques « qui dépendent de senseurs fournissant des séries de données en temps-réel où, pour atteindre un traitement rapide et compact, leurs parties avioniques sont programmées en assembleur – un type de programmation qui ne se prête pas à la construction de vérifications[28] ».

Tous ces arguments ont pour originalité de pointer l'insuffisance de la vérification formelle au regard de la *matérialité* de la machine qui interagit avec le monde, et on les appelle ainsi parfois *arguments de matérialité*[29]. Ils remettent spécifiquement en cause le second pare-feu logique de la programmation, celui des erreurs de fonctionnement. Pour Fetzer, un ordinateur est un dispositif causal matériel. En droit toute mathématique de ce système ne peut être que conjecturelle, devant être validée par l'expérience – ou plutôt non falsifiée, avance Fetzer dans une référence à Popper[30]. Un programme est ainsi une conjecture concernant le réel, de même qu'un théorème de mathématiques appliquées vise un système matériel par le biais d'une interprétation soumise à l'inspection empirique inductive[31].

§ 49. Le succès des méthodes formelles

D'une certaine manière, l'histoire a plutôt donné tort à ces critiques. Les méthodes formelles se développent de manière spectaculaire depuis les années 1990.

27. (Fetzer 1988, p. 1057).
28. (ibid., p. 1062).
29. Voir par exemple (Varenne 2019).
30. (Fetzer 1988, p. 1062).
31. (ibid., p. 1059-60).

Plusieurs articles[32] documentent la dynamique certaine de cette approche depuis une trentaine d'années. Les méthodes formelles ne participent certes qu'à une faible part des projets de développement, mais elles sont devenues légitimes et respectées grâce à leurs succès dans la programmation de systèmes embarqués critiques, par exemple de transport ou de télécommunications. On peut citer, à Paris, le logiciel de contrôle de la ligne 14 automatisée du métro, qui a été entièrement développé grâce aux méthodes formelles.

Ces succès s'appuient sur plusieurs progrès techniques. En particulier, les premiers langages de spécification comme Z ou VDM permirent de modéliser la logique du problème à traiter et de vérifier la cohérence de sa formulation, d'en prouver des propriétés préliminaires, d'en explorer la combinatoire. Il s'agit là, comme le disait Smith, d'un bénéfice majeur des méthodes formelles. L'automatisation de la vérification, qui peut être partielle ou totale, selon la technique utilisée, mit à mal les arguments pragmatiques contre sa possibilité concrète. Quant au fait que les besoins évoluent fréquemment, les méthodes formelles apparaissent à cet égard plutôt comme un avantage, car des incohérences ou des oublis introduits par une nouvelle spécification peuvent être immédiatement détectés : c'est dans la maintenance du logiciel, et sur le long terme, que les méthodes formelles prouvent leur utilité.

Les succès des méthodes formelles dans les systèmes embarqués apportent également des réponses aux arguments de matérialité contre les méthodes formelles, comme ceux de Fetzer que nous venons de voir. En effet la démarche de vérification peut être itérative, et s'appliquer à la conception même de l'ordinateur, puisque celui-ci est aussi constitué d'une certaine organisation de circuits logiques inspectables mathématiquement. C'est *de facto* dans de tels projets de conception de machines que les méthodes formelles eurent leurs premiers succès. L'idée sous-jacente fut très bien anticipée dès 1969 par Tony Hoare, un allié ferme de Dijkstra dans cette croisade en faveur des méthodes formelles :

> Lorsque la correction d'un programme, de son compilateur et du matériel de l'ordinateur aura été établie avec une certitude mathématique, il sera possible de placer une grande confiance dans les résultats du programme, et de prédire leurs propriétés avec une confiance seulement limitée par la fiabilité de l'électronique [...] Les avantages pratiques de la vérification des programmes l'emporteront finalement sur les difficultés, compte tenu du coût croissant des erreurs de programmation[33].

32. (Butler et al. 2020 ; Fitzgerald et al. 2013 ; Mashkoor, Kossak et Egyed 2018).
33. (Hoare 1969, p. 580).

Dans le cas de figure anticipé par Hoare, le risque d'erreurs de fonctionnement serait négligeable, puisqu'il n'y aurait plus alors que l'électronique elle-même, lors de l'exécution du programme, qui pourrait décevoir les attentes du maître d'ouvrage. Si l'ensemble des composants matériels ont été eux-mêmes vérifiés avec la même rigueur, « alors on n'a plus qu'à se préoccuper de rayons cosmiques égarés, et de choses similaires[34] » . L'expression *rayons cosmiques* est un *topos* ironique récurrent des partisans des méthodes formelles. Contre leurs adversaires qui énoncent l'impossibilité de droit qu'une machine matérielle se comporte *toujours* comme un modèle abstrait le prédit, le rayon cosmique représente les facteurs causaux externes imprévisibles, mais d'une probabilité infiniment faible, qui pourraient invalider la certitude quasi-logique concernant une propriété matérielle.

Le problème aujourd'hui pour que soit réalisée à grande échelle la vision de Hoare – celle d'une chaîne d'exécution entièrement vérifiée selon des méthodes formelles, programme, compilateur, système d'exploitation, architecture matérielle, composants électroniques, etc. – ne se situe pas au niveau des « couches basses » mais plutôt à celui du compilateur, pourtant l'une des motivations principales de cette approche à ses débuts. On s'aperçut en effet assez vite que, bien qu'on fût là dans un domaine formel par excellence (traduire automatiquement les textes d'un langage formel à un autre) de nombreuses ambiguïtés et complexités étaient possibles quant au comportement souhaité de la machine pour telle ou telle instruction. La vérification des compilateurs reste, aujourd'hui encore, un des domaines les plus complexes en informatique, notamment du fait de la difficulté à tout simplement spécifier formellement un langage de haut niveau : la spécification de Java, par exemple, fait actuellement près de 800 pages. Cependant une telle tâche n'est pas impossible, comme le montre le développement d'un compilateur pour le langage C, CompCert[35].

3.2 L'insuffisance de l'interprétation formelle

Cette section est consacrée à montrer l'insuffisance de l'interprétation de la programmation comme activité formelle. Nous commençons par remettre en cause la conception naïve de la distinction entre spécification et programme sur laquelle elle repose (§ 50), que nous illustrons ensuite à l'aide de trois exemples de projets

34. (Bringsjord 2015, p. 275).
35. (Leroy 2016).

de programmation de types courants. Le cas des projets ERP* montre que toute spécification est bi-face, c'est-à-dire qu'elle prescrit également des changements précis dans l'environnement du programme, notamment quant au comportement attendu de ses utilisateurs (§ 51). La difficulté qu'il y a à contrôler le bon comportement de calculs concurrents, et plus généralement à contrôler des composants physiques de systèmes embarqués, montre que la programmation est, d'une manière essentielle, tournée vers la matérialité de la machine, ce qui n'exclut pas les approches formelles, bien au contraire (§ 52). Enfin, les projets d'EDI* montrent la possibilité d'une imbrication complète des problématiques d'implémentation et de spécification (§ 53).

§ 50. Spécification et programme

Comme on l'a vu, Dijkstra interprète la spécification comme un pare-feu logique entre le travail du maître d'ouvrage et celui du programmeur. Cette vision ne correspond pas à la réalité des projets de programmation, où ces travaux apparaissent imbriqués l'un dans l'autre. Nous montrons deux formes possibles de cette imbrication. La première résulte du succès même des méthodes formelles* et conduit à la *dérivation* directe du programme à partir de la spécification. La seconde, issue des méthodes agiles*, place au centre du projet la collaboration et la *négociation* entre maître d'ouvrage et programmeur.

Un des progrès majeur des méthodes formelles fut leur prolongement à la programmation elle-même. Nous avons vu que l'étape de conception pouvait être décomposée en deux activités, celle de conception proprement dite, qui détermine l'algorithme et les structures de données nécessaires à la résolution de problème, et celle du codage de cette solution dans un langage de programmation donné.

On s'aperçut assez vite que la distance entre une spécification assez détaillée et un programme qui la satisfaisait pouvait être, dans bien des cas, assez ténue. Lorsque la solution algorithmique était simple, il n'était pas difficile de l'inclure dans la spécification elle-même, de telle sorte que celle-ci devenait complètement explicite, ne formulant pas seulement le problème mais également sa solution. La correspondance entre un tel type de spécification et le programme devenait alors si direct qu'il était possible d'automatiser le passage de l'une à l'autre. Avec Event-B, le langage qui fut utilisé pour la ligne 14 du métro, la solution algorithmique elle-même est décrite logiquement, sa cohérence interne est prouvée, ainsi que son accord avec la spécification. Le code lui-même peut alors être généré automatique-

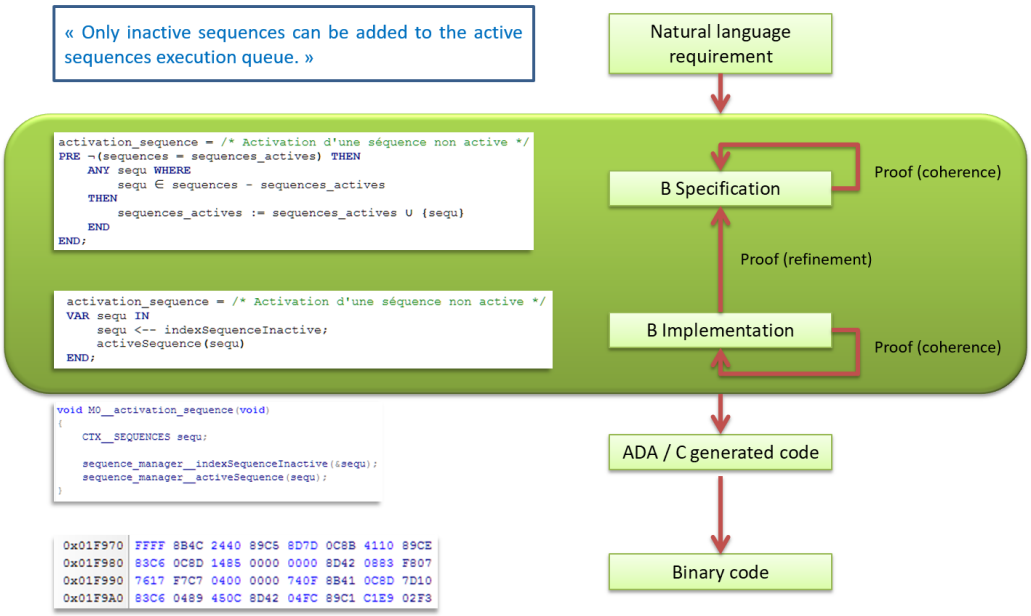


FIGURE 3.3 – Illustration du processus de développement en Event-B
Source : (CLEARSY 2021)

ment. La figure 3.3 décrit ce processus de développement automatisé : d'abord, l'expression de besoins en langage naturel, puis la spécification, ensuite l'« implémentation », qui signifie ici la description de la solution algorithmique, enfin la génération de code ADA ou C, qui ne sont ici que des intermédiaires pragmatiques de compilation*.

Ces approches « cassent » en quelque sorte l'interprétation classique de la programmation, puisqu'un bloc essentiel apparaît (sur fond vert dans la figure) qui regroupe l'étape de spécification et celle de la conception algorithmique. C'est là que se fait l'essentiel du travail du programmeur, c'est-à-dire dans la description logique de son problème et de sa solution. Cela contredit la vision de Dijkstra, pour qui l'étape de spécification se situe en amont du travail du programmeur. Par ailleurs l'écriture du programme lui-même, la deuxième activité de l'étape de conception, est ici repoussée à ce que nous avons appelé l'implémentation, c'est-à-dire la préparation de la machine à l'exécution du programme.

L'interprétation des méthodes formelles doit donc être modifiée ainsi que représenté dans la figure 3.4. Ces approches ne sont pas cantonnées aux méthodes formelles. Ainsi, dans le domaine de la programmation-objet, on développa dès le milieu des années 1990, en complément du langage de spécification UML, des

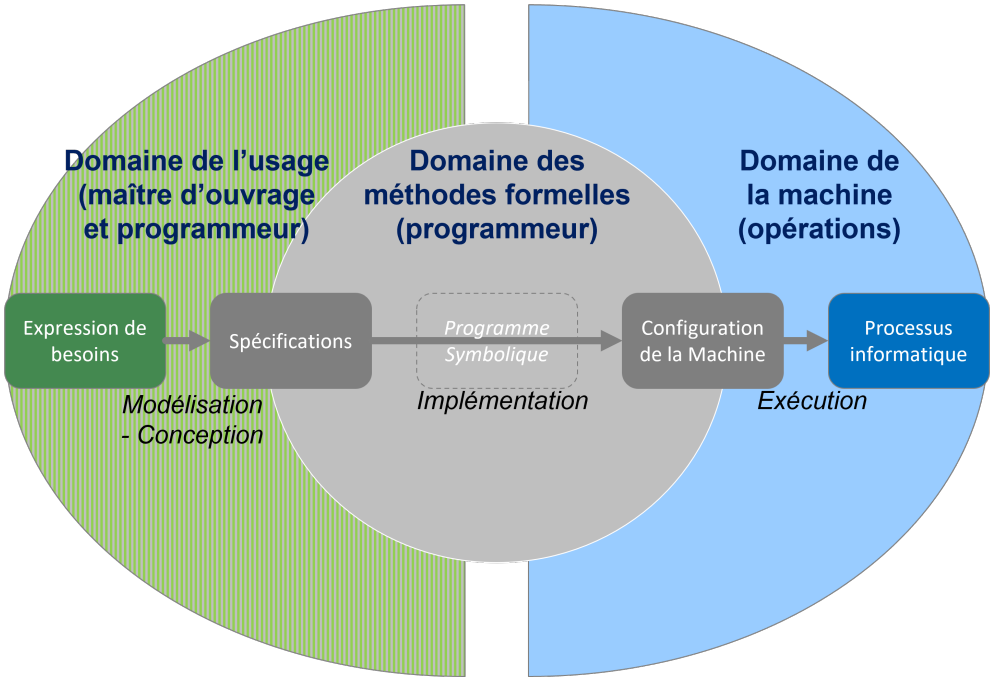


FIGURE 3.4 – Interprétation de la programmation dans le schéma Event-B

générateurs automatique de code à partir de ses diagrammes de classe, d'état ou de séquence[36]. Ces outils sont aujourd'hui intégrés à une approche plus vaste du développement logiciel appelée *ingénierie pilotée par les modèles* (*model driven engineering*) affirmant le primat du modèle (autrement dit, l'objet de la spécification) sur le programme[37].

Les méthodes formelles ont eu également un effet indirect, celui d'augmenter la robustesse des langages de programmation eux-mêmes. HOARE (1996, p. 11-13) en cite trois : la programmation structurée*, le typage des données* et le masquage des données*. Le langage **Eiffel**, sur lequel nous revenons plus loin, va plus loin et inclut des possibilités d'intégrer la vérification au code de manière souple[38], insérant ainsi l'approche des méthodes formelles au cœur même des pratiques habituelles du programmeur.

Cette tendance a permis l'émergence d'une autre forme d'imbrication entre spécification et programme, où ils sont conçus comme résultat d'une collaboration et d'une discussion entre maître d'ouvrage et programmeur. En effet, lorsque des lan-

36. Voir (MUKHTAR et GALADANCI 2018) pour un panorama sur ce sujet.
37. Voir (KLEIN, LEVINSON et MARCHETTI 2015) pour un panorama sur cette tendance.
38. (MEYER 2008).

gages permettent de documenter de manière rigoureuse les conditions de validité des programmes, l'écriture de spécifications formelles devient moins importante. *De facto*, elle peut alors ressembler à un simple dédoublement du programme dans un langage seulement un peu plus abstrait. Y a-t-il alors vraiment une valeur à l'écrire de deux manières différentes de manière à pouvoir s'en assurer plus formellement, comme lorsqu'on pose de deux manières différentes un calcul ? mais si c'est le cas, on ne voit plus très bien ce qu'il y aurait d'essentiel à cette manière de faire ?... De fait il est souvent déclaré que :

> Le programme lui-même est la seule description complète de ce qu'il fera[39].

ou encore :

> Pour de nombreux systèmes, le code du programme est la seule spécification précise de ce que fait le système[40].

Dans ces cas de figure, la spécification se confond avec une expression de besoins détaillée, et on insiste pour qu'elle soit écrite en langage naturel, même si elle doit suivre un certain formalisme. Les méthodologies agiles* recommandent de travailler directement sur le programme lui-même plutôt que sur la documentation, ainsi que l'exprime clairement le *Manifeste agile* :

> Nous valorisons […] des logiciels opérationnels plus qu'une documentation exhaustive[41].

Dans les projets suivant ces méthodologies, les maîtres d'ouvrage et les développeurs travaillent ensemble sur des phases courtes de développement appelés sprints*, typiquement quinze jours, ayant des objectifs clairs définis dès le départ. Dans ce cadre, la formalisation détaillée des besoins se fait au tout début du sprint*, et s'améliore de manière itérative tout du long, par le dialogue quotidien entre maîtres d'ouvrage, utilisateurs, et développeurs, sous l'égide de l'assistant à maître d'ouvrage qui rédige ce cahier des charges[42]. La communication est centrale dans cette approche, comme l'exprime clairement le *Manifeste agile* :

> La méthode la plus simple et la plus efficace pour transmettre de l'information à l'équipe de développement et à l'intérieur de celle-ci est le dialogue en face à face[43].

On peut dans ce cadre se demander s'il est bien nécessaire d'écrire des spécifications lorsque le logiciel va être développé à peu près au même moment, et que les

39. P.J. Davis, cité par (De Millo, Lipton et Perlis 1979)
40. (Hayes et C. B. Jones 1989).
41. (Beck et al. 2001).
42. Voir par exemple (Lothon 2012).
43. (Beck et al. 2001).

deux documents doivent converger ensemble. Or c'est justement peut-être là que la valeur spéciale de la spécification apparaît. Outre qu'elle permet de documenter les discussions de façon à éviter des désaccords ou des oublis, un motif objectif est, dans le cas où le programmeur et le maître d'ouvrage appartiennent à deux entités juridiques distinctes, *le besoin de contractualisation et de conformité*[44]. Pour les juristes, il est important de garder trace de ce qui a été vraiment demandé par un client à son fournisseur, afin de mieux pouvoir formaliser un contrat et mieux départager les responsabilités en cas de problème. Nous revenons sur ce point essentiel plus loin.

La programmation, dans les recommandations agiles*, doit donc plutôt se représenter ainsi que sur la figure 3.5. Le programme se situe au confluent de deux interactions collaboratives : l'une avec le maître d'ouvrage, que nous venons d'expliciter, et une seconde qui favorise, sur le modèle de la première, la mise en production* rapide des développements, grâce une collaboration accrue entre équipes de développement et d'opérations*, un modèle qui donne lieu à la démarche Devops*.

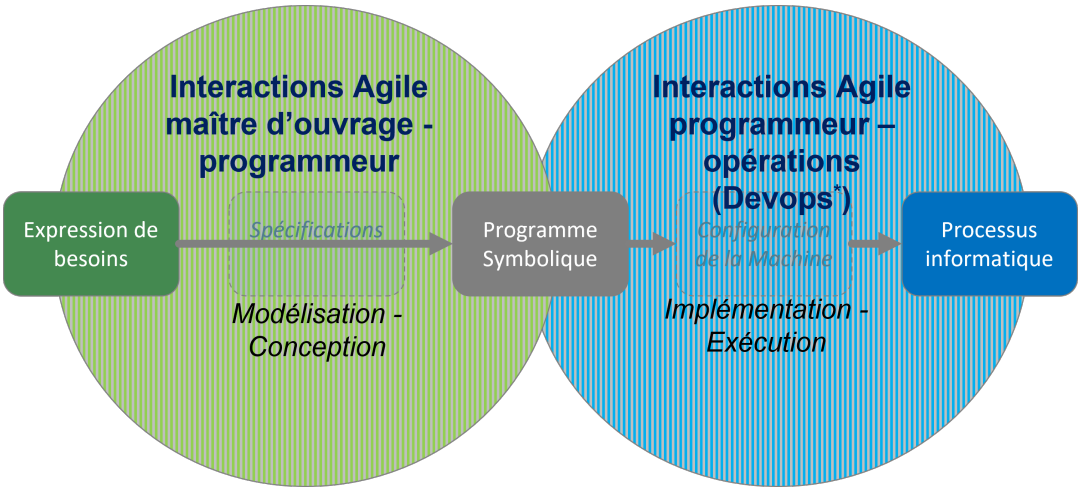


FIGURE 3.5 – Interprétation de la programmation selon les méthodes agiles

Qu'on suive donc les méthodes formelles ou les méthodes agiles* (les deux ne sont d'ailleurs pas incompatibles) on parvient au même constat : c'est une fiction de l'esprit que d'imaginer le programmeur occupé uniquement d'algorithmique formelle et de traductions entre langages symboliques de haut et de bas niveau,

44. (ANAND 2021).

dans une double indifférence au problème réel que son programme va permettre de résoudre, et à la machine qui va l'exécuter. L'interprétation de la programmation exposée par la figure 3.2 n'a rien à voir avec les méthodes formelles. Elle décrit simplement mal la division logique du travail qui a lieu dans les projets de programmation. Le maître d'ouvrage, le programmeur et le responsable des opérations ne sont pas des travailleurs postés qui transforment successivement des produits intermédiaires depuis une expression de besoins jusqu'à un processus informatique. Cette fiction peut sembler convaincante seulement si on s'occupe de problèmes qui sont *déjà* symboliques, où l'expression de besoins est déjà clarifiée et où l'exécution devrait se dérouler sans problème. Mais dans le cas général, elle ne tient pas. C'est ce que nous allons tenter de montrer à présent à l'aide de trois exemples.

§ 51. Les projets ERP et la gestion du changement

Le premier exemple concerne la mise en place de grands progiciels d'entreprise, et notamment les ERP* (*Enterprise Resource Planning*) qui permettent une gestion intégrée de l'information au sein d'une organisation, grâce à la centralisation des bases de données produites et consommées par de multiples services (comptabilité, ventes, production), leur évitant ainsi à construire des interfaces d'échanges entre eux, source d'erreurs et de ralentissement.

Les ERP, offrant dans une suite logicielle unique l'essentiel des besoins de traitement de données d'une entreprise, ont connu un succès majeur dans les grandes et moyennes entreprises au cours des années 1990-2000. Ils ont souvent remplacé, dans les grandes organisations, des centaines de petites applications développées localement, difficiles à maintenir et à interfacer entre elles. Une seule grande plateforme détient ainsi une vue cohérente de toutes les données (finance, logistique, achats, ventes…) et elle interface entre eux les départements. Par exemple, une commande commerciale émise par le service client génère automatiquement un bon de sortie de stock à la logisitique, un ordre de production à l'usine et une facture à la comptabilité. La maintenance en est simplifiée : si le modèle de données doit être mis à jour, par exemple modifier la structure de la fiche client, cela ne doit être fait qu'une seule fois, et non sur plusieurs applications, ce qui limite d'autant les risques d'erreurs. Le marché des ERP est aujourd'hui concentré autour de quatre ou cinq éditeurs dominants, dont SAP et Oracle[45].

45. Pour une introduction historique aux ERP, voir (Jacobs et Weston 2007 ; Pollock et

Un des avantages des ERP, étant donné la domination de ces quelques éditeurs, est de capitaliser sur leur expérience cumulée. Par exemple, un besoin fonctionnel exprimé par de multiples organisations va sans doute être intégré à une prochaine version du logiciel, dans un cadre qui assure sa cohérence avec l'ensemble des autres règles métier*. Aussi ces logiciels arrivent-ils équipés des modèles des processus métier les plus courants, qui reflètent le plus souvent les meilleures pratiques du domaine en question, et que l'organisation n'a plus qu'à personnaliser. SAP R/3, en particulier, lancé en 1992, a été un moment décisif dans l'industrie. Comme on peut le voir dans la figure 3.6, son modèle couvre tous les processus clés de la finance, des achats, des ventes et de la distribution et de la production. SAP a également développé des « verticales » pour traiter les processus particuliers d'industries spécifiques, telles que la notation du crédit dans le secteur bancaire. Cela a créé des normes *de facto* pour les procédures d'entreprise, auxquelles les organisations ont dû largement s'adapter.

Le déploiement d'ERP change souvent en profondeur le fonctionnement d'une organisation. Ce qui était informel devient tout à coup objet de règles précises. L'application des règles s'impose *de facto*, puisque le système se bloque si des actions « hors processus » sont menées. Par exemple l'obligation de déclarer un bon de commande lorsqu'on souhaite acheter une nouvelle imprimante empêche l'utilisateur de choisir un modèle et un fournisseur qui ne sont pas dans la base de données. Si on ne le fait pas, le fournisseur ne sera pas mis en paiement par l'entreprise, à moins de créer une exception dans l'ERP, un droit réservé à des personnes autorisées. Cet exemple montre, plus profondément encore, que les objets construits par l'organisation (bon de commande, facture, etc.) voient leur effectivité transférée à l'informatique, en ce que *ce qui fait foi* à présent est ce qui est enregistré dans le système, et non plus l'engagement personnel d'un employé à effectuer une action – par exemple un document manuscrit de sa part ou une affirmation orale.

Ainsi cette révolution de l'informatique d'entreprise a également révolutionné le fonctionnement interne des organisations. C'est tout autant le progiciel qui est adapté à l'entreprise que celle-ci qui doit s'adapter à lui. Cela est manifeste lorsqu'on étudie les facteurs de succès de la mise en place d'un tel système. Un tel projet est toujours un défi ; il dure de 6 mois à 2 ans, et peut coûter jusqu'à 50 millions de dollars américains pour de très grandes entreprises. Par exemple, un petit projet pour une entreprise de 150 salariés a été évalué à 2 millions de dollars

Williams 2008).

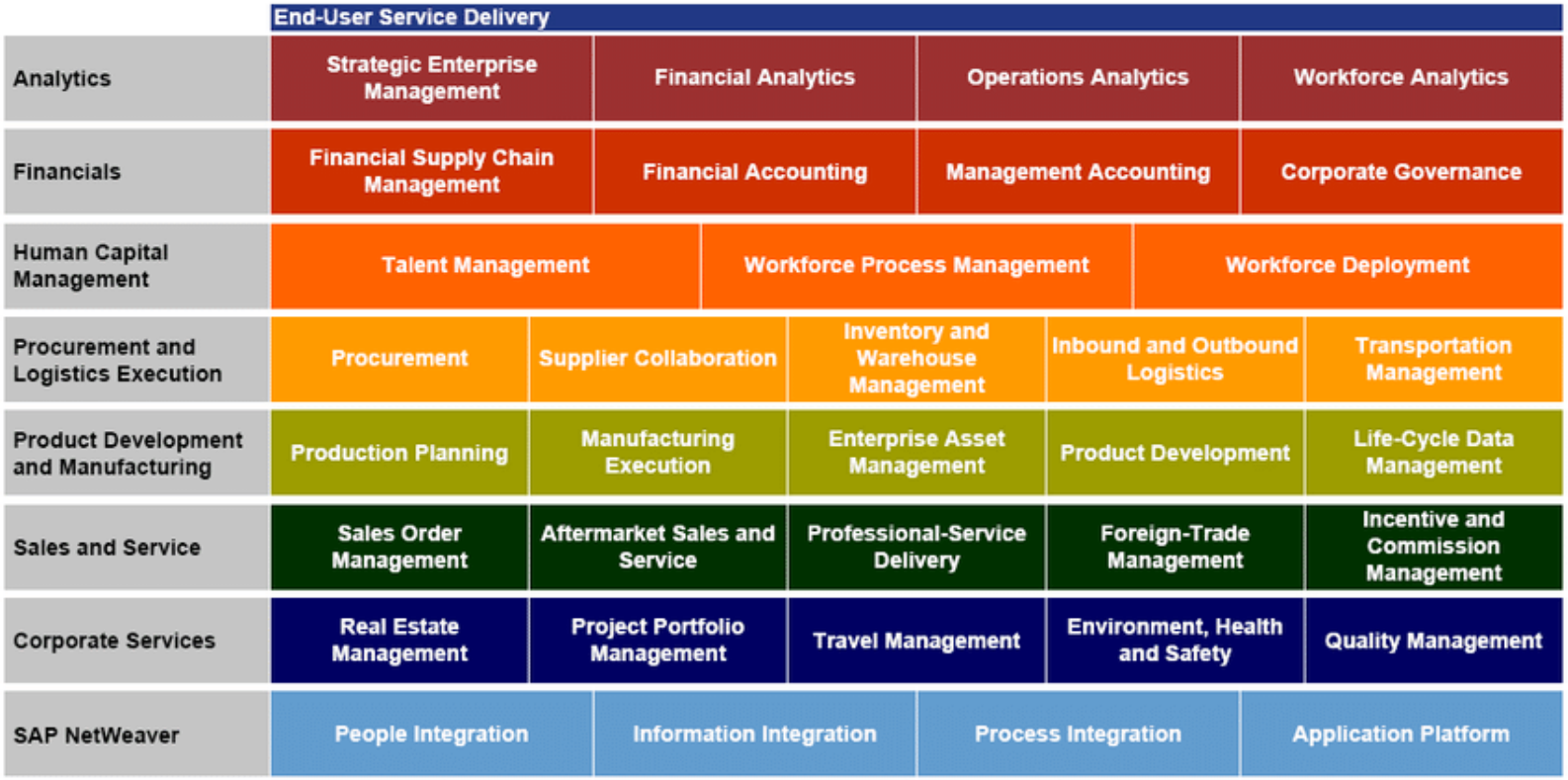


FIGURE 3.6 – La carte de référence des domaines de processus définis dans le logiciel SAP R/3
Source : v3it.com

en 2019[46]. Par ailleurs le risque d'échec de ces projets est également élevé.

La complexité de ces projets ne réside pas dans leur partie logicielle : celle-ci est extrêmement robuste. Leur problème se situe au niveau des interactions entre l'informatique et l'organisation. Concernant l'implémentation, de nombreuses interfaces doivent être construites entre l'ERP et les systèmes environnants, qu'ils soient internes à l'entreprise (par exemple un logiciel de production d'une usine) ou externes (par exemple le système de facturation d'un client). Ces interfaces nécessitent de s'assurer de la compatibilité des formats des données échangées entre systèmes. C'est pour cela que la mise en place d'ERP prend souvent le nom d'*intégration*.

Du côté des utilisateurs et de la maîtrise d'ouvrage, un écart important peut exister entre les processus de l'entreprise et ceux prédéfinis par l'ERP, qui doit être cartographié soigneusement dans la phase d'expressions de besoins. Cet écart peut en général être résolu par la paramétrisation du logiciel, qui est très modulaire et dont la plupart des processus sont configurables. Par exemple, il est aisé de choisir s'il faut une ou deux validations pour une transaction donnée, et d'indiquer quelles personnes peuvent l'effectuer. Ou encore, il est aisé de différencier la priorité d'une production ou d'un flux de transport en fonction du type de produit ou de client à servir.

Cependant, de manière régulière, il se trouve que des processus de l'entreprise ne s'alignent pas aisément avec le référentiel du logiciel. Il y a alors deux options principales (avec de nombreuses variantes intermédiaires) : développer des programmes qui vont permettre de « contourner » l'ERP concernant la fonctionnalité en jeu, ou adapter le fonctionnement de l'organisation au standard proposé par l'ERP. La première solution est coûteuse et risquée d'un point de vue informatique, car elle compromet justement la cohérence interne du logiciel. Aussi, à l'exception de processus critiques, c'est souvent la seconde solution (ou une variante intermédiaire) qui est choisie[47].

Lorsque l'ampleur des changements est conséquente, notamment en termes de nombre d'utilisateurs impactés, on accompagne la mise en place du nouveau logiciel par *une gestion du changement*. Outre la formation des utilisateurs, on met en place des dispositifs de mobilisation des équipes : pédagogie quant à la nécessité de ce changement, explication des bénéfices attendus pour l'organisation mais aussi

46. Expérience personnelle de l'auteur.
47. (Soh, Kien et Tay-Yap 2000 ; Soh et Sia 2004).

pour chaque employé, visibilité sur la durée du changement, sur les perturbations occasionnées, attitudes à adopter face aux clients, etc. Cette gestion du changement est l'un des facteurs de coût parmi les plus importants de ces projets, mais aussi un élément essentiel de leurs succès.

La mise en place d'un ERP est un cas de programmation qui ne coïncide pas avec le modèle classique pour deux raisons. Avant de les mettre en lumière, il faut bien insister qu'il s'agit bien là d'une programmation au sens où nous entendons ce terme[48]. Même si l'écriture de code y est souvent très limitée, puisque le logiciel existe déjà, toutes les autres étapes d'un projet de programmation sont bien présentes : expression de besoins, spécification, mise en production, tests, validation. Le fait que le développement du logiciel parte d'un cadre pré-programmé n'est pas essentiel pour notre propos. D'abord, on y trouve toujours un peu de codage, du fait de spécificités à prendre en compte. Par ailleurs les paramétrisations du logiciel peuvent être si complexes qu'elles s'apparentent à de la programmation. Notamment, les requêtes SQL qui y sont très présentes sont bien des programmes. Enfin, même dans le cas d'un ERP qui partirait de zéro, une gestion de changement chez les utilisateurs est nécessaire, puisqu'un tel logiciel force en tous les cas l'homogénéisation de pratiques et réduit les marges de manœuvre des utilisateurs.

Les projets d'ERP mettent en évidence les défauts de l'interprétation formelle de la programmation. La prescription ne va pas unilatéralement de l'expression de besoins au programme. Au contraire, le logiciel lui-même peut modifier non seulement l'expression de besoins, mais le comportement des utilisateurs eux-mêmes. De ce fait, l'implémentation matérielle du logiciel doit être accompagnée d'une gestion humaine du changement qui en est le corrélat indissociable. C'est autant le comportement des personnes que celui des machines qui déterminent le succès global du projet. Le logiciel, pour fonctionner proprement, impose un certain comportement à ses utilisateurs ; on peut même argumenter que la mise en place du logiciel a souvent pour objectif principal de standardiser et de contrôler ce comportement, au-delà des bénéfices informatiques déjà mentionnés[49].

Ce fait illustre remarquablement la thèse qui semblera évidente aux praticiens, que la mise en place d'un logiciel est autant une affaire de préparation et de transformation de la situation avec laquelle il interagira, que de développement du logiciel. L'expression de besoins aussi bien que l'implémentation ont un carac-

48. Voir plus haut, § 21.
49. Ce point sera développé au chapitre 5.

tère « bi-face » : il ne s'agit pas uniquement pour les humains de prescrire des comportements à la machine, mais également de laisser la machine dicter certains comportements aux personnes humaines.

§ 52. Les calculs parallèles de haute performance

Sans même considérer la gestion du changement dans l'organisation, l'implémentation matérielle des ERP présente également des défis techniques importants. Implémenter une interface n'est pas seulement une question de configuration d'échanges de données (un problème que nous abordons au développement suivant) mais également de synchronisation de systèmes fonctionnant en temps réel, ou en tous les cas selon des logiques temporelles souvent incompatibles. Par exemple, les vieux systèmes fonctionnent souvent en traitement par lot* (*batch processing*), c'est-à-dire qu'ils n'acceptent de recevoir les données que par de larges fichiers, à des fréquences fixes, avant de les traiter et les renvoyer toutes ensembles, ce qui n'est pas toujours compatible avec des systèmes traitant leurs données au fil de l'eau. Il peut donc y avoir autre chose que des « rayons cosmiques » qui perturbent un système par ailleurs tout à fait bien vérifié formellement, à savoir des contraintes liées à l'interactivité ou au parallélisme des processus.

Il n'est pas besoin d'un ERP pour observer ce phénomène. L'exemple très simple d'une addition réalisée de manière distribuée* permet de le montrer. Considérons les trois nombres binaires suivants[50].

$$a = 1.10000 \quad b = 0.0000011 \quad c = 0.0000001$$

Les nombres fractionnels sont codés en général en deux parties, l'une donnant l'exposant (c'est-à-dire la première puissance non nulle du nombre, ici 0 pour a, puisque le premier 1 se trouve un chiffre avant la virgule et que $10^0 = 1$, -6 pour b, et -7 pour c, puisque le premier 1 se trouve respectivement à 6 et 7 chiffres après la virgule) et l'autre la mantisse, c'est-à-dire le nombre divisé par sa première puissance. Cette dernière est codée sur un nombre de bits prédéterminé, ce qui fait que les mantisses très longues, dépassant cette limite, sont tronquées. Si on code les nombres ci-dessus avec des mantisses de 6 bits, celles de a et b seront 1.10000, et celle de c 1.00000.

50. Nous empruntons cet exemple à (Ludwig 2018).

Cette méthode de codage remet en cause l'associativité de l'addition. Supposons qu'on souhaite additionner les nombres ci-dessus $a + b + c$:

$$\begin{array}{rl} a & 1.10000 \\ +\,b & 0.0000011 \\ \hline = a+b & 1.1000011 \end{array}$$

On voit ici que, la mantisse de $a+b$ étant limitée à 6 bits, les deux derniers chiffres 11 (en rouge) seront tronqués, et que la mantisse de $a+b$ vaudra 1.10000, comme a. De la même manière, si on ajoute c à ce résultat, celui-ci restera inchangé, car l'exposant de c est trop faible pour que sa valeur soit prise en compte. On a donc

$$(a+b)+c = 1.10000$$

Cependant si on additionne d'abord b et c, on a :

$$\begin{array}{rl} b & 0.0000011 \\ +c & 0.0000001 \\ \hline = b+c & 0.0000100 \end{array}$$

Si on ajoute ce résultat à a, on a alors :

$$\begin{array}{rl} a & 1.10000 \\ +(b+c) & 0.00001 \\ \hline = a+(b+c) & 1.10001 \end{array}$$

Ainsi $(a+b)+c \neq a+(b+c)$ du fait de la troncature des longues mantisses. Dans le cas d'une distribution parallèle d'un tel calcul, si a, b et c sont trois résultats intermédiaires, calculés parallèlement, qui viennent incrémenter un même registre de mémoire, ce dernier peut donc voir sa valeur dépendre de la vitesse relative à laquelle chacun des trois calculs a été réalisé. Or cette vitesse relative est en général indéterminée, dépendant sans doute elle-même de calculs antérieurs et de l'état des différents processeurs. Ces erreurs de troncature ne sont absolument pas négligeables dans de nombreuses applications scientifiques. Un modèle climatique peut effectuer 10^{14} opérations par seconde sur 100 000 processeurs parallèles. Ces calculs étant itératifs, les erreurs minimes se propagent et se composent rapidement, quelle que soit la longueur de la mantisse.

Ainsi, dans ce cas de figure, la programmation ne peut nullement se passer de considérer la manière dont les calculs qu'elle spécifie vont en pratique être réalisés par la machine, et elle doit, si elle veut s'assurer de leur validité, contrôler l'aspect matériel de leur exécution, par exemple en indiquant des priorités, des points de synchronisation, etc. On pourrait croire que ces techniques de « bas niveau » sont gérées au niveau des librairies* des compilateurs*, et qu'elles devraient être donc stables. Cela est bien le cas, mais seulement au niveau d'une version donnée d'un compilateur donné. Comme c'est justement au niveau de ces techniques que se situent les progrès concernant l'optimisation numérique, celles-ci évoluent de manière régulière. Le progrès technique lui-même rend donc les résultats des modèles non réplicables d'une version à l'autre. La figure 3.7 montre les écarts importants entre deux exécutions d'un même modèle climatique, suite à la mise à jour d'une libraire optimisant la durée de calcul. La première ligne de la figure montre le gain de temps réalisé de la version **O2** à **O3** d'une simulation climatique. On voit cependant dans les graphiques que les deux exécutions, quoique se ressemblant globalement, diffèrent fortement point par point. Ce problème de reproductibilité des simulations est un enjeu majeur pour les sciences climatiques, car il fournit des arguments aux militants climato-sceptiques.

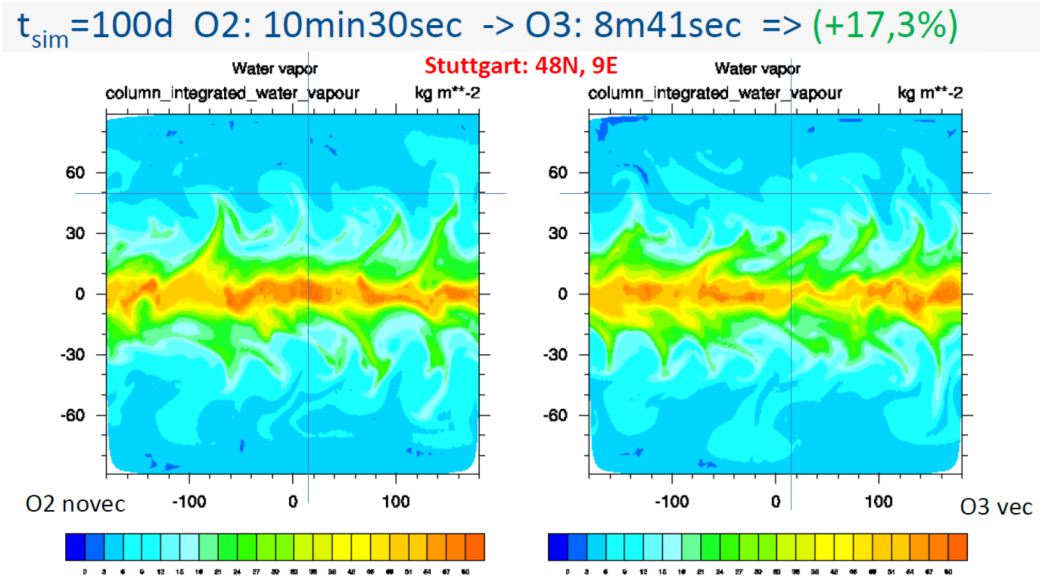


FIGURE 3.7 – Deux exécutions d'un même modèle climatique avec deux versions **O2** et **O3** d'une librairie d'optimisation (Source : (Ludwig 2018).

D'un certaine manière, ce résultat peut sembler évident. La programmation vise à contrôler le comportement *d'une machine*, et donc c'est en référence à elle qu'elle a d'abord une signification. L'interprétation formelle, orientée vers la programmation de fonctions mathématiques dont la calculabilité est parfaitement bien définie, oublie ce fait fondamental.

Ce fait est encore plus visible dans les cas où le programme a explicitement pour fonction de contrôler une machine intervenant de manière causale dans l'environnement physique, qu'on appelle couramment *systèmes embarqués* ou *systèmes cyber-physiques*. Par exemple, le contrôleur d'un portillon pour une entrée payante doit se bloquer dès qu'il a été activé et qu'il n'y a plus de crédit dans le système de paiement. *« Dès que »* n'est cependant pas une spécification précise. En observant que le système mécanique ne peut être à nouveau disponible qu'au bout de 750 ms, on peut spécifier cette durée comme borne supérieure : le contrôleur doit re-bloquer l'accès *au plus tard* 750 ms après une entrée. Cette spécification a l'avantage d'éviter de bloquer le portillon au cas où un crédit est à nouveau enregistré dans cet intervalle[51].

La programmation ne peut nullement ici faire abstraction *de* la machine, au contraire c'est celle-ci même dont elle vise à modéliser le comportement de manière adéquate de telle sorte qu'on puisse mieux la contrôler. Cette particularité est bien exprimée par Lee (2017, p. xi) dans leur ouvrage sur ces systèmes :

> Lorsqu'on étudie les systèmes cyber-physiques, certains problèmes importants apparaissent, qui sont rares dans l'informatique générale. [Là], par exemple, le temps qu'il faut pour effectuer une tâche est une question de *performance*, et non de *correction*l [...Au contraire,] dans les systèmes cyber-physiques, le temps requis par une tâche peut être critique pour le bon fonctionnement du système. Dans le monde physique, contrairement au monde cybernétique, le passage du temps est inexorable.
> De plus, […] les processus physiques sont des compositions de plusieurs choses qui se déroulent à la fois, contrairement aux processus logiciels, qui sont profondément enracinés dans des étapes séquentielles. […] Dans le monde physique, en revanche, les processus sont rarement procéduraux [mais] des compositions de nombreux processus parallèles. Mesurer et contrôler leur dynamique en orchestrant des actions qui les influencent sont les tâches principales des systèmes embarqués. […] De nombreux défis techniques découlent de la nécessité de relier une sémantique intrinsèquement séquentielle à un monde physique intrinsèquement concurrent.

Tout cela ne rentre pas non plus dans le cadre de l'interprétation formelle puis-

51. (M. Jackson 2010, p. 155).

qu'ici ce sont des sujet d'implémentation qui semblent s'inviter dans l'expression de besoins et la spécification. La machine, c'est-à-dire ce qui exécutera le processus informatique programmé, ne peut être négligée dans la conception du programme.

Cette double porosité – que les utilisateurs du programme doivent également coopérer à son bon fonctionnement, et que la machine physique soit partie de la spécification – pointe vers un troisième fait, qu'il n'y a pas toujours de séparation nette entre les usagers du programme et la machine qui l'exécute.

§ 53. La complexité des connexions EDI

Ce fait se voit particulièrement bien dans le cas de la programmation d'échanges de données entre deux systèmes informatiques. Les grands logiciels d'entreprise sont en interconnexion permanente avec d'autres logiciels, internes ou externes à l'entreprise. Le problème consiste à préciser les formats d'échange des données entre logiciels, la signification des différents libellés structurant les données ainsi que leur codification. Par exemple des produits peuvent être codés différemment entre fournisseur et client, il faut alors développer une *trans-codification*.

Pour faciliter ce travail dans le cas de transactions régulières, une famille de protocoles a été définie, l'EDI*, qui permet d'établir un échange régulier de données transactionnelles entre deux entreprises ou deux sites. L'EDI définit des normes pour différents types de transactions, par exemple l'envoi d'un bon de commande, d'un avis d'expédition, d'une facture, d'une feuille de paie, etc. La figure 3.8 donne l'exemple d'une conversation EDI entre le système d'un vendeur et d'un acheteur : bon de commande, récépissé du bon, bon de livraison, facture. Les données transmises prennent la forme de fichiers texte qui suivent une nomenclature bien définie. Le code 3.1 montre le fichier EDI d'un bon de commande type. Le fait qu'il s'agit d'un bon de commande se voit ligne 3, grâce au code `850` qui y figure. La date de la commande figure ligne 2 (`20101127`). La ligne 8, avec le code `DTM*002*` indique que la date de livraison requise est le 14/12/2010. L'adresse de l'acheteur se trouve lignes 12-14. Lignes 15-32 se trouvent les commandes de six références de snacks, occupant 3 lignes chacune. Un récapitulatif occupe les lignes 33-34, avec le nombre de références commandées et le montant total de la transaction. Les trois dernières lignes sont des indicateurs de fin du message qui renvoient à des codes d'ouverture au début.

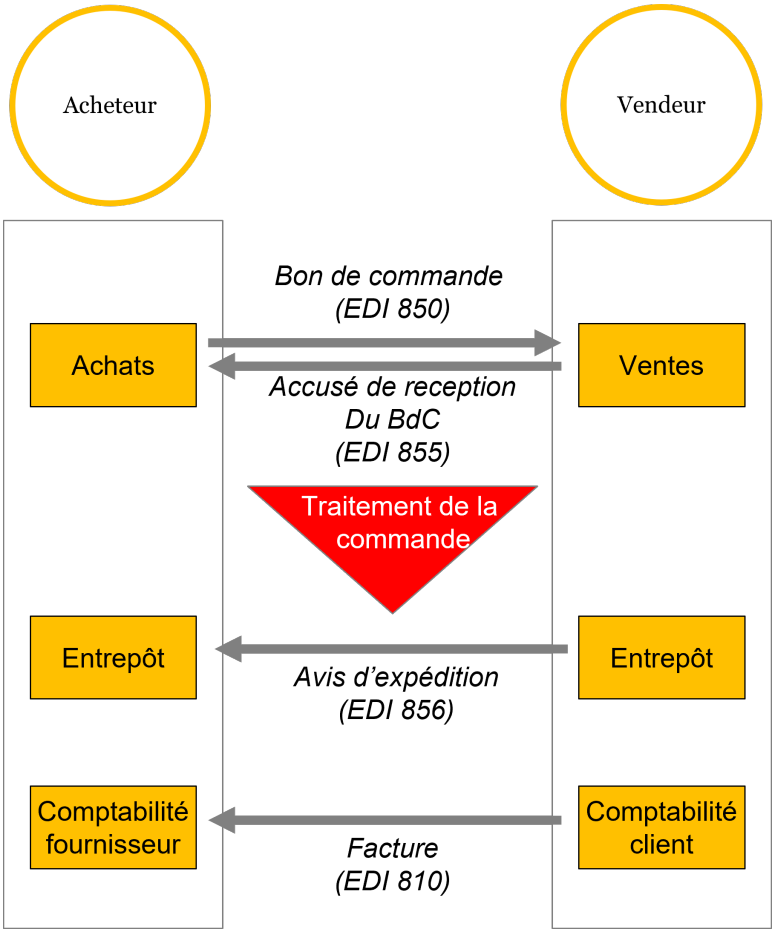


FIGURE 3.8 – Principe de fonctionnement du standard EDI

CODE 3.1 – Fichier EDI : modèle d'ordre d'achat de marchandises, Source : www.1edisource.com/resources/edi-transactions-sets/edi-850

```
1  ISA*01*0000000000*01*0000000000*ZZ*ABCDEFGHIJKLMNO*ZZ
       *123456789012345*101127*1719*U*00400*000003438*0*P*>
2  GS*PO*4405197800*999999999*20101127*1719*1421*X*004010VICS
3  ST*850*000000010
4  BEG*00*SA*08292233294**20101127*610385385
5  REF*DP*038
6  REF*PS*R
7  ITD*14*3*2**45**46
8  DTM*002*20101214
9  PKG*F*68***PALLETIZE SHIPMENT
10 PKG*F*66***REGULAR
```

```
TD5*A*92*P3**SEE XYZ RETAIL ROUTING GUIDE
N1*ST*XYZ RETAIL*9*0003947268292
N3*31875 SOLON RD
N4*SOLON*OH*44139
PO1*1*120*EA*9.25*TE*CB*065322-117*PR*RO*VN*AB3542
PID*F****SMALL WIDGET
PO4*4*4*EA*PLT94**3*LR*15*CT
PO1*2*220*EA*13.79*TE*CB*066850-116*PR*RO*VN*RD5322
PID*F****MEDIUM WIDGET
PO4*2*2*EA
PO1*3*126*EA*10.99*TE*CB*060733-110*PR*RO*VN*XY5266
PID*F****LARGE WIDGET
PO4*6*1*EA*PLT94**3*LR*12*CT
PO1*4*76*EA*4.35*TE*CB*065308-116*PR*RO*VN*VX2332
PID*F****NANO WIDGET
PO4*4*4*EA*PLT94**6*LR*19*CT
PO1*5*72*EA*7.5*TE*CB*065374-118*PR*RO*VN*RV0524
PID*F****BLUE WIDGET
PO4*4*4*EA
PO1*6*696*EA*9.55*TE*CB*067504-118*PR*RO*VN*DX1875
PID*F****ORANGE WIDGET
PO4*6*6*EA*PLT94**3*LR*10*CT
CTT*6
AMT*1*13045.94
SE*33*000000010
GE*1*1421
IEA*1*000003438
```

L'intérêt de ces nomenclatures est leur caractère standardisé, qui permet à chaque entreprise de n'avoir à développer qu'un seul traducteur de fichiers EDI pour intégrer les données reçues à son système, de quelque autre entreprise qu'elles proviennent.

Cependant il est des cas où des standards n'existent pas ou sont insuffisants : il faut alors que les deux parties s'entendent sur un format d'échange. Cela est par exemple le cas dans l'industrie ophtalmique où, certains verres devant être fabriqués sur mesure, ce ne sont pas des références produit qui doivent être échangés, mais des paramètres géométriques, qui peuvent varier d'un fabricant à l'autre, et qui évoluent sans cesse du fait de l'innovation technique.

Selon l'interprétation formelle de la programmation, il suffirait que les deux parties partent des spécifications de chacun des deux systèmes concernant les don-

nées à échanger pour en déduire la spécification de la conversion, qui peut alors faire l'objet d'un contrat explicite précisant les travaux et les responsabilités de chaque partie. Ensuite chacune développerait leur convertisseur dans le format d'échange agréé, et elles testeraient enfin ensemble le résultat, toutes ces choses étant bien sûr itératives.

Dans la réalité, de telles spécifications sont illusoires, car les systèmes qu'il faut connecter ont souvent des défauts relativement à leurs spécifications idéales, du fait même de leur évolution constante, qui ne peuvent être corrigées dans le cadre d'un simple projet d'interfaçage. Par exemple leurs référentiels de données peuvent se révéler incomplets (tel paramètre n'étant pas prévu dans l'un des systèmes), ou organisés différemment (ainsi si plusieurs produits considérés comme une référence unique d'un côté sont distingués de l'autre), ou encore certaines valeurs de données peuvent avoir des sens spéciaux (un cas classique est celui des champs vides pouvant signifier `0` ou valeur inconnue, ce qui n'est pas la même chose), ou parfois des abréviations sont utilisées mais non documentées, etc.

Le travail d'interface se fait donc heuristiquement et très souvent par tâtonnements. À chaque problème rencontré, une négociation s'engage pour savoir quelle partie va devoir adapter son format de données à l'autre. Chaque système, tel qu'il est *implémenté*, avec ses défauts, devient ainsi *de facto* la spécification des travaux à réaliser par l'autre partie, celle-ci devant « tordre » ses propres données pour les rendre compatibles. Le projet de développement se réduit alors à un travail très largement empirique de tâtonnements et de tests en masse*. On a tenté de représenter cette situation par la figure 3.9, où les flèches grises dénotent le fonctionnement idéal du projet, et les flèches rouges sa réalité : la machine de chaque partie devient la spécification de l'autre, par ajustements empiriques.

On pourrait rétorquer que cette situation très particulière, si elle ne correspond pas au modèle, ne le remet pourtant pas en cause. Ce n'est pas en effet parce qu'on n'arrive pas à se conformer à un idéal que celui-ci est nécessairement caduc.

Cependant, nous pensons que l'EDI peut nous mettre sur la piste d'un modèle plus général concernant le type de médiation que le programme réalise entre l'utilisateur et la machine. D'abord, des machines tierces peuvent être à l'origine d'une partie des spécifications du programme à réaliser. Elles doivent alors être comprises comme faisant partie de l'environnement externe de la machine qui exécutera le programme, au même titre que des utilisateurs. On peut penser que cette possibilité est évidente, puisque cela ne change rien pour une machine que ses

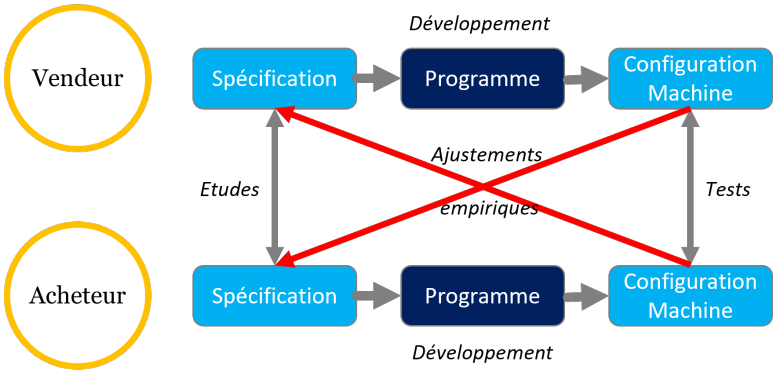


FIGURE 3.9 – Fonctionnement idéal *vs* réel d'un projet EDI

interlocuteurs soient des personnes ou d'autres machines : elle n'est sensible qu'à l'échange de données, et non à la nature essentielle de leurs sources. Mais il y a ici plus que cela : il s'agit de machines qui vont elles-mêmes être l'objet d'un travail de programmation parallèle, qui est critique pour le succès du projet global. On se retrouve dans une situation identique à celle d'un projet ERP, où la *gestion du changement* chez les utilisateurs (personnes humaines) est requise pour que l'implémentation soit un succès. En d'autres termes, l'environnement du programme doit lui-même faire une partie du chemin pour que le programme fonctionne correctement. Ce qu'on voit là, c'est que cette modification de l'environnement peut n'être rien d'autre qu'*une autre programmation*. Qu'est-ce qui permet alors de la considérer comme *hors projet*, de ne pas s'en soucier ? C'est justement qu'elle soit déterminée à l'avance comme se situant au-delà du périmètre de la machine dont on a la responsabilité. C'est cette notion centrale de *responsabilité* que nous allons à présent approfondir.

3.3 L'interprétation contractuelle de la programmation

Dans cette section, nous mettons en avant le rôle central de la spécification dans la séparation des responsabilités entre maître d'ouvrage et programmeur, notamment grâce à un cadre de réflexion proposé par l'informaticien Michael Jackson (§ 54). Nous illustrons ensuite ce cadre conceptuel par un exemple concret de programmation donné par Jackson, concernant le contrôle d'un portillon d'accès payant (§ 55). Dans le développement § 56 nous en montrons la portée générale en

le rapprochant d'autres considérations pragmatiques et logiques, afin d'esquisser sur cette base une interprétation nouvelle de la programmation, que nous appelons *contractuelle*. Nous la mettons à l'épreuve en la confrontant aux études de cas exposées à la section précédente (§ 57).

§ 54. Qu'est-ce qu'une spécification ?

Au développement § 50, nous avions remis en cause le besoin de distinguer *spécification* et *programme*, montrant que, selon les approches de la programmation, l'une prenait le pas sur l'autre, mais qu'il semblait redondant d'en dupliquer l'élaboration.

Contre ces doutes, le philosophe Ray Turner entreprend une défense vigoureuse du concept de *spécification*, tout en reconnaissant qu'il ne se distingue pas du concept de *programme* par des propriétés formelles observables. Les spécifications peuvent être détaillées ou floues, formelles ou informelles, explicites ou implicites ; elles peuvent être écrites dans un langage formel, voire exécutables grâce aux générateurs de code. *Tout* peut valoir comme spécification, du moment qu'on lui *donne une valeur législatrice sur la correction ou non du programme évalué*. La seule caractéristique d'une spécification est son rôle normatif :

> En général, ce que nous considérons comme spécification et ce que nous considérons comme artefact ou programme n'est pas une question de forme linguistique, ni que l'un soit écrit dans un langage de spécification qui n'a pas d'implémentation, et l'autre écrit dans un langage de programmation qui en a une. De telles caractérisations semblent passer à côté du point conceptuel le plus important : quelque chose est une spécification quand elle possède la *juridiction* de correction sur un artefact[52]. *(Nous soulignons)*

La spécification joue ainsi un rôle logique semblable à celui d'une définition stipulative[53] : elle décrit un concept de ce qui doit ou peut exister, sans expliciter nécessairement comment le construire ; elle permet de juger de l'adéquation à ce concept d'objets ou d'artefacts candidats à sa norme.

La spécification est donc tout entière dans ce geste stipulatif. Elle établit une norme à travers laquelle on peut juger un programme. Doit-on comprendre cette norme comme une loi ou un règlement, qui serait émise par le maître d'ouvrage ? Une telle unilatéralité n'est pas très réaliste, on l'a vu dans le cas du projet ERP.

52. (Turner 2011, p. 147).
53. (ibid., p. 139).

Par ailleurs une loi énonce un cas général, et en général les spécifications servent à juger la conformité d'un objet précis. Il semble bien plutôt que le concept juridique adéquat pour comprendre le rôle des spécifications est celui de *contrat*, qui établit un partage local de responsabilité entre deux parties mutuellement consentantes. C'est donc grâce aux spécifications que le contour des responsabilités du programmeur, ainsi que son périmètre d'action, sont établis. *De facto*, les spécifications, ou encore le *cahier des charges*, sont des objets contractuels essentiels dès qu'il s'agit d'une fourniture de produit ou de service – nous avions déjà mentionné ce fait à propos des méthodes agiles*. Comme le disent Gunter et al., une des motivations importantes de la spécification est « la nécessité de répartir les responsabilités dans un contrat entre les besoins de l'utilisateur et du fournisseur. Ils construisent leur accord autour de S [la spécification], qui leur sert de base de communication[54]. » Ce rôle médiateur des spécifications est central dans la réflexion de l'informaticien Michael Jackson qu'il expose avec sa collègue Pamela Zave dans plusieurs articles qu'un ouvrage collectif[55] rassemble partiellement et met en perspective. Un des contributeurs à cet ouvrage considère ce travail si important pour la discipline de l'expression de besoins et de la spécification qu'il compare la formule centrale de Jackson et Zave, $E, S \vdash R$, sur laquelle nous reviendrons, à celle d'Einstein $E = mc^2$ [56].

Le point de départ de tout accord entre un maître d'ouvrage et un programmeur est d'opérer une distinction nette entre le *domaine du problème* et le *domaine de la machine* :

> Le premier principe fondamental de la description consiste à identifier clairement ce qui est décrit. Dans le développement logiciel, la distinction principale se situe entre la machine et le domaine du problème. La machine est constituée par le logiciel que nous construisons et l'ordinateur qui l'exécute. Le domaine du problème est la partie du monde dans laquelle la machine doit produire l'effet souhaité par le client pour qui le système est développé. Par exemple […] dans le système de commande d'un ascenseur, le domaine du problème est constitué par la cage d'ascenseur, la cabine d'ascenseur, le moteur et le mécanisme d'enroulement, les boutons, l'éclairage, les étages desservis par l'ascenseur et les personnes qui l'utilisent[57].

Cette formulation permet d'emblée de percevoir que l'expression de besoins ne

54. (Gunter et al. 2000, p. 40).
55. (Nuseibeh et Zave 2010).
56. (A. Hall 2010).
57. (M. Jackson 2002, p. 5).

concerne pas d'abord le logiciel lui-même, mais avant tout un « état du monde » qu'on souhaite modifier, un problème réel qu'on souhaite résoudre. Le domaine du problème est le lieu logique où à la fois s'expriment les besoins que le programme doit réaliser mais également les propriétés pertinentes du monde dont il devra tenir compte, notamment les contraintes et les modalités d'action des différents moyens à disposition. M. Jackson, qui se méfie des ambiguïtés liées à l'opposition descriptif / prescriptif souvent utilisée dans ces discussions, préfère simplement indiquer que les propriétés invariantes doivent être exprimées par des propositions ayant une modalité *indicative* : elles expriment ce qui est, tout simplement, sans condition. Les besoins, par contre, doivent être exprimés par des propositions ayant une modalité *optative* : ils expriment ce qu'on souhaiterait voir advenir.

Une note terminologique est ici nécessaire. Dans ce texte, M. Jackson utilise l'expression *domaine du problème* pour désigner l'ensemble des phénomènes pertinents à l'expression des invariants et des besoins du problème. Il la remplace ailleurs par le terme d'*environnement*, et le plus souvent par celui de *monde* (*world*) de manière très générique. Tous ces termes ont des défauts pour la réflexion que nous souhaitons mener à partir de l'idée sous-jacente. *Domaine du problème* est trop abstrait et malcommode. *Environnement*, au contraire, semble restreindre l'idée à des circonstances matérielles, tandis qu'un domaine de problème est également décrit par des relations structurelles. *Monde* vise trop large, car il est seulement question ici des faits et des relations *pertinents* pour un problème donné. Le concept recherché doit donc avoir une portée seulement locale. Comme l'expression de M. Jackson a elle-même varié, nous nous autorisons à utiliser le terme *situation* qui nous semble le mieux éviter les défauts relevés et correspondre assez bien à l'idée visée par M. Jackson. Une situation peut décrire des phénomènes environnants concrets pour la considération d'un problème, mais également des relations de structure entre ces phénomènes. Une situation est toujours locale, cependant elle permet d'évoquer l'ouverture et l'indétermination relative qu'on trouve souvent dans les problèmes concrets, une connotation certainement recherchée par M. Jackson à travers le terme de *monde*.

> Il est essentiel de distinguer clairement entre les propriétés du domaine de problème qui sont données, et celles que la machine doit réaliser. Les propriétés données sont celles que le domaine du problème possède, quel que soit l'effet de la machine. Le fait que l'ascenseur ne puisse aller du deuxième au quatrième étage sans passer par le troisième ne dépend pas du comportement de la machine. Il est intégré à la structure du domaine. […] Les besoins, en revanche, ne sont pas des propriétés données,

> mais souhaitées. Ce sont des contraintes supplémentaires qui doivent être réalisées par le comportement approprié de la machine à son interface avec le domaine du problème. C'est une confusion grave que de faire une seule description du domaine du problème, en combinant les propriétés du domaine avec les besoins. Il est difficile de savoir, à partir d'une description combinée, si une propriété particulière est un objectif du développement ou une hypothèse sur laquelle on peut s'appuyer pour atteindre les objectifs[58].

La confusion que déplore Jackson dans le dernier paragraphe du texte est en effet fréquente. Il n'y a pas, dans la pratique courante de la modélisation, de distinction nette entre propriétés et besoins de la situation. Pour le voir, on peut consulter la typologie des exigences auxquelles doit prêter attention le maître d'ouvrage dans sa phase de collecte, telle que l'établit un ouvrage de référence sur la question :

1. Les objectifs [comme] par exemple *« diminuer de 20 % le temps d'attente au guichet »* […] sont exprimés, dans la majorité des cas, par le donneur d'ordre. Le verbe sous-entendu est le verbe vouloir […] : *« nous (l'entreprise) voulons diminuer le temps d'attente […] »*
2. Les cas d'utilisation ou scénarios correspondent à des besoins des utilisateurs et […], par exemple *« enregistrer un nouveau client »*. La phrase sous-entendue est avoir besoin de, exprimée à la première personne : *« J'ai besoin d'enregistrer tout nouveau client. »*
3. Les règles sont des exigences réglementaires ou des règles métier (également appelées règles de gestion). Le verbe sous-entendu ou explicite est devoir […] Par exemple : « *« Un retrait d'espèces ne peut être effectué que si le compte est positif. »* […]
4. Les exigences fonctionnelles constituent le cœur et souvent la partie la plus importante d'un cahier des charges. Elles expriment un comportement requis de la part du système. Elles dérivent des catégories précédentes. Par exemple : *« Si le retrait d'espèces n'est pas autorisé, le système envoie un message au client. »*
5. Les exigences de qualité, également appelées exigences non fonctionnelles, bien qu'elles n'en constituent qu'une partie. Elles s'expriment sous forme d'adjectif (rapide, facile...)
6. Les exigences d'interface qui expriment le besoin d'une communication entre le système à l'étude et le monde extérieur : matériel, logiciel et personnes.
7. Les contraintes techniques, comme l'utilisation d'un système ou d'un langage

58. (M. Jackson 2002, p. 7).

particulier, ou de technologies spécifiques, comme un protocole de communication ou de sécurité[59].

On serait tenté de classer les *objectifs, les besoins, les exigences fonctionnelles et non fonctionnelles* comme les besoins proprement dites (modalité optative), tandis que *les règles métier, les exigences d'interface, et les contraintes techniques* seraient plutôt partie de ses propriétés (modalité indicative).

Cependant cette distinction ne tient pas. Une propriété (par exemple une règle de TVA) peut aussi être perçue comme un besoin à satisfaire : tout dépend de savoir s'il est de la responsabilité du programme à concevoir de calculer la TVA ou si celle-ci est calculée dans un programme séparé avec lequel s'interfacer.

Nous revenons plus loin sur ce caractère *mobile* de la frontière entre propriété invariante et besoin, qui se fonde elle-même sur celle entre machine et situation. Pour l'instant il importe de bien cerner le caractère contractuel de la spécification. Jackson dénomme E (pour *Environment*) l'ensemble des propriétés de la situation qui sont tenues pour invariantes, et par R (*Requirements*) les besoins qu'on voudrait y voir réalisés. Les spécifications S sont alors un groupe de propositions qui portent sur des observations possibles dans les deux domaines, celui du problème et celui de la machine. Si les propositions S sont vérifiées, comme le sont par définition les propriétés E, alors les besoins R doivent être satisfaits. M. Jackson écrit donc la relation suivante, qui affirme qu'on doit pouvoir prouver R si E et S sont admis :

(3.1) $E, S \vdash R$

On voit ici que R et S jouent un rôle logique entièrement différent. La spécification n'est pas la réécriture des besoins en un langage plus précis. Elle décrit un ensemble de phénomènes qui, s'ils se trouvent réalisés, entraînent la réalisation des phénomènes décrits par R. Ici il n'est pas dit comment interpréter cette relation, s'il s'agit de phénomènes causaux, volitifs, logiques, mathématiques, etc. Cela dépend du problème posé. Par exemple la spécification suivante : *« S'il y a du crédit dans le portillon, celui-ci doit pouvoir être activé »* réalise le besoin suivant : *« Si un visiteur a payé, il doit pouvoir entrer au zoo »*. On voit bien que l'implication de la seconde proposition par la première suppose des propriétés de la situation qui n'ont rien à faire avec le programme : par exemple que, quel que soit le moyen

59. (Constantinidis 2015, p. 108). Nous omettons les deux derniers types qui sont des variantes de contraintes techniques.

de paiement du client, ce paiement soit acheminé au système de crédit du portillon, qu'activer le portillon permet d'entrer, que ce portillon donne bien accès au zoo et non aux toilettes. Il s'agit là d'obligations qui restent de la responsabilité du maître d'ouvrage du programme, et qui doivent être aussi réalisées afin que ses besoins soient satisfaits.

Une seconde différence entre spécifications et besoins est que les premières ne peuvent porter que sur des phénomènes accessibles à la machine. *Accessible* ici veut dire observable ou contrôlable – car la machine doit pouvoir agir sur la situation présente. À l'opposé, les besoins qui décrivent un problème ne sont pas nécessairement situés à cette interface – par exemple le fait qu'un client entre dans le zoo après avoir activé le portillon n'est pas un phénomène observable par la machine qui contrôle le portillon. On peut seulement espérer qu'un événement contrôlé par la machine puisse le faire advenir.

Le rôle du programmeur est alors de réaliser les spécifications S grâce à la machine. Celle-ci peut être décrite par un ensemble de propriétés invariantes M de la machine qui, conjointes aux relations établies par le programme P, réalisent S. On a alors la formule similaire à la précédente 3.1 :

(3.2) $M, P \vdash S$

(Nous précisons ici que ces formules sont des simplifications de relations logiques plus complexes qui permettent d'assurer par exemple la cohérence des besoins R ou des spécifications S. L'article de Gunter et al. (2000) précise le modèle formel complet adéquat. Nous nous contentons ici des formules simplifiées, suffisantes pour notre propos.)

On peut noter encore que M et P peuvent porter sur des phénomènes qui sont « cachés » à l'environnement, c'est-à-dire qui ne doivent pas lui importer. Par exemple, le programme peut, afin de résoudre une tâche, stocker et modifier une certaine donnée dans la mémoire de la machine ; le bon fonctionnement du programme peut être exprimé par les valeurs prises par cette donnée, mais cela est indifférent à la spécification et encore plus à l'expression de besoin, car d'autres solutions pourraient être envisageables. Jackson dit que ces phénomènes sont cachés à la situation, ou encore, ce qui nous semble préférable, internes à la machine.

Ces deux formules permettent d'exprimer les deux parties du contrat. La formule 3.1 traduit que la réalisation de la spécification S suffit au maître d'ouvrage, puisque la validité des propriétés E de la situation sont également de son ressort,

et qu'ensemble elles réalisent ses besoins R. Quant à la formule 3.2, elle traduit bien sûr les engagements du programmeur, de fournir un programme P et une machine M qui réalisent les spécifications S.

Gunter et al. (2000) représentent ce cadre de réflexion par la figure 3.10 (légèrement modifié), qui montre bien la portée de ces différents éléments. Ce schéma

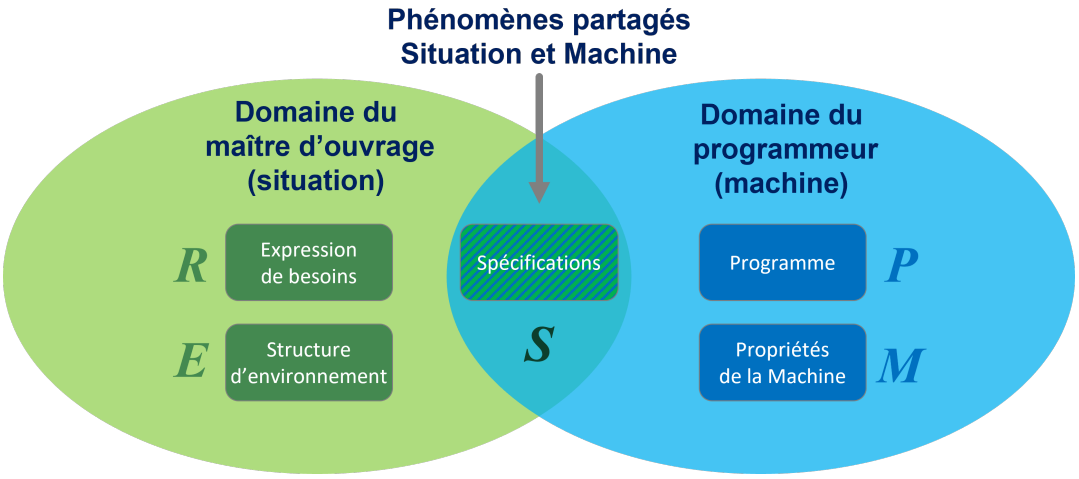


Figure 3.10 – Interprétation de la programmation selon Michael Jackson (légèrement modifiée)

ne représente pas les mêmes choses que les précédents. D'abord, il ne prétend pas représenter la dynamique de développement elle-même (conception, modélisation, etc.) Il s'agit d'un schéma statique, qui représente tous les objets à construire (ou déjà construits). Ensuite, tous ces objets sont de même nature : il s'agit de descriptions (dans l'idéal, formulées par des clauses logiques) de propriétés des domaines considérés, valant inconditionnellement (la machine, l'environnement) ou devant advenir (le programme, la spécification, les besoins).

§ 55. Un exemple : un portillon d'accès

Afin de rendre plus concret ce cadre de réflexion, il est utile de reprendre systématiquement un exemple déjà évoqué dans ce chapitre, que développent M. Jackson (2010), celui du portillon d'entrée au zoo. On suppose que le besoin du maître d'ouvrage (le directeur du zoo) est simple : 1) Il ne faut pas laisser entrer de visiteurs s'il n'y a plus de crédit dans le système de paiement du portillon. 2) Il faut laisser entrer un visiteur lorsqu'il y a du crédit. Il faut noter qu'il n'est

pas demandé de vérifier l'identité du payeur et du visiteur (un accompagnateur de groupe scolaire pouvant payer pour plusieurs enfants), et qu'il doit être possible de créditer le système de plusieurs entrées avant que celles-ci soient effectuées (paiement et entrée ne doivent pas nécessairement alterner).

Ces besoins sont spécifiés de la manière suivante. La situation peut être observée par la machine grâce à deux paramètres : le crédit total collecté par le système, le nombre total de fois où le portillon a été activé. On peut remarquer ici que l'activation du portillon, qui est observable par la machine, tient lieu de l'entrée qui ne l'est pas. Ce point doit être agréé entre le maître d'ouvrage et le programmeur, car il se pourrait par exemple qu'un client active le portillon et se ravise avant d'entrer. Cette situation n'est pas prévue par la spécification.

Si cette clause est agréée, on obtient deux premières spécifications : 1) le crédit total collecté doit être toujours supérieur ou égal au nombres d'activations. 2) S'il est strictement supérieur (et seulement dans ce cas ci), le portillon doit pouvoir être activé. Cette étape de la réflexion n'a fait que traduire les besoins du client en termes observables par la machine.

La spécification progresse ensuite en remplaçant ces conditions par d'autres qui font intervenir des paramètres *contrôlables* par la machine. Or celle-ci peut simplement bloquer ou débloquer le portillon. Il faut donc traduire les spécifications 1 et 2 selon ces paramètres.

La condition 1) peut être assurée par les deux conditions suivantes : 1a) Tant que le crédit total reçu et le nombre total d'activations sont égaux, il ne faut pas débloquer le portillon. 1b) *Dès qu'ils deviennent égaux*, alors il faut bloquer le portillon. Si la condition 1) est initialement vérifiée (par exemple au début de la journée les deux compteurs sont à zéro), alors 1a) et 1b) assurent qu'elle le reste toujours.

De manière symétrique, la condition 2) est traduite ainsi : 2a) Tant que le crédit total reçu est supérieur au nombre total d'activations, il ne faut pas bloquer le portillon. 2b) *Dès qu'il devient* supérieur, alors il faut débloquer le portillon.

La spécification n'est pas encore terminée, car les conditions 1b) et 2b), qui décrivent des réactions à des événements, ne sont pas assez précises : il faut également indiquer quels délais sont acceptables pour ces réactions. Pour la condition 1b), nous avons déjà évoqué la justification du délai maximal de 750 ms plus haut[60], qui permet d'éviter le resquillage. Pour la condition 2b) il s'agit de débloquer le

60. Voir plus haut, § 52.

portillon aussi vite que possible afin de ne pas faire patienter le visiteur qui a payé, et un délai de 250 ms est proposé.

La programmation de cette spécification est simple et n'est pas étudiée par Jackson et Zave. On peut écrire le programme correspondant à la spécification comme suit, en langage Python :

CODE 3.2 – Programme Python répondant à la spécification du contrôleur de portillon

```
1    While True:
2        v = info_verrou()
3        c = info_credit()
4        p = info_active()
5        if not(v) and c==p:  cmd_verrouiller()
6        if v and c > p: cmd_deverouiller()
```

Ce programme est constitué d'une boucle infinie introduite par la ligne 1 : le programme itère sans cesse sur lui-même pour vérifier l'état du portillon et agir si besoin. Chaque itération se passe comme suit : le programme récupère d'abord (lignes 2-4) auprès des composants matériels du portillon son état de verrouillage puis les valeurs du crédit total et du nombre total d'activations, et les place respectivement dans trois variables `v` (un booléen*), et `c,p` (des nombres entiers). Les lignes 5 et 6 implémentent respectivement les clauses 1b et 2b de la spécification.

Il n'est pas nécessaire d'implémenter les clauses 1a et 2a, puisqu'elles indiquent seulement des événements qui ne doivent pas arriver. Il s'agit d'invariants qu'on doit pouvoir prouver formellement (à condition, encore une fois, que la condition 1 soit initialement vérifiée). Cependant, il pourrait y avoir une faille logique et matérielle dans le programme si, entre deux itérations, le portillon pouvait être activé deux fois ou plus, de telle sorte que le nombre d'activations, initialement inférieur d'une unité au crédit total, se retrouve strictement supérieur à lui à l'itération suivante. Dans ce cas on aurait `c<p`, une condition qui n'est pas testée par le programme, et qui de plus viole la condition 1.

La possibilité d'une telle faille nous amène aux clauses de délai décrites dans la spécification. Elles n'apparaissent pas dans le programme car elles concernent la machine, qui doit être suffisamment réactive pour que le programme soit exécuté dans les durées indiquées. L'exécution du code Python est sans doute négligeable, mais la récupération d'informations de dispositifs externes à la machine, opérée par les fonctions `info_xxx`, et les commandes qui leurs sont transmises, opérées

par les fonctions `cmd_xxx` peuvent avoir une certaine latence. Si $dmax()$ est la mesure de la durée maximale d'exécution de ces fonctions, il faut que la machine satisfasse les clauses suivantes :

$$(3.3)\quad dmax(\text{info\_verrou}()) + dmax(\text{info\_credit}()) + dmax(\text{info\_active}()) + dmax(\text{cmd\_verrouiller}()) \leq 750 \text{ ms}$$

$$(3.4)\quad dmax(\text{info\_verrou}()) + dmax(\text{info\_credit}()) + dmax(\text{info\_active}()) + dmax(\text{cmd\_deverrouiller}()) \leq 250 \text{ ms}$$

Ces clauses permettent de montrer que le portillon ne peut être activé deux fois de suite entre deux itérations. Si 750 ms est la durée minimale d'une activation, il suffit en effet que :

$$(3.5)\quad dmax(\text{info\_verrou}()) + dmax(\text{info\_credit}()) + dmax(\text{info\_active}()) \leq 1500 \text{ ms}$$

Or il s'agit d'une condition qui est toujours vérifiée du fait de 3.3, et encore plus de 3.4. On a par là la garantie qu'entre deux itérations du programme la valeur de `info_active()` ne peut être incrémentée que d'une seule unité au plus. Dans ce cas, la clause `c≥p` est toujours gardée valide par le programme.

Enfin, pour achever la complétude de l'analyse, il faut au niveau le plus élémentaire établir des clauses qui concernent les fonctions et les commandes utilisées par le programme (de la forme `info_xxx` et `cmd_xxx`). D'abord, qu'elles font bien ce qu'elles sont censées faire, autrement dit par exemple que le compteur d'activations du portillon fonctionne bien et que la fonction correspondante enregistre bien sa valeur lorsqu'elle est sollicitée, mais également que la machine a le monopole du verrouillage ou du déverrouillage du portillon. Si tel n'est pas le cas, une faille logique pourrait s'infiltrer dans le fonctionnement du programme. Ces clauses peuvent être considérées comme partie de la machine M ou de l'environnement E selon le domaine de responsabilité du programmeur. C'est avec toutes ces clauses qu'on peut démontrer que le programme proposé permet de réaliser le besoin du directeur du zoo.

Cet exemple montre la progression logique de la programmation. La résolution du problème procède par des reformulations progressives en des termes observables, puis contrôlables par la machine. Dans ce processus, des hypothèses sont

faites quant à la situation, par exemple que chaque activation correspond à une entrée au zoo, que le portillon est correctement initialisé, etc. La spécification est terminée lorsque l'on peut démontrer que, si les propositions spécifiant les actions de contrôle de la machine sont respectées, ainsi que les hypothèses concernant la situation, alors les besoins du maître d'ouvrage sont satisfaits.

Il est entendu que ce processus de résolution est une reconstruction logique de l'investigation, et non sa démarche réelle, qui a pu emprunter d'autres chemins et a sans doute été itérative. A. Hall (2010) remarque que, dans ces exemples, tous les termes de la solution sont donnés dès le départ, ce qui est peu réaliste. C'est l'analyse des besoins du client qui a sans doute suscité l'idée de mettre en place les deux compteurs mentionnés, ainsi qu'un mécanisme de verrouillage du portillon. La démarche exposée par Zave et Jackson n'expose absolument pas comment on en est venu à *cette* solution, parmi toutes les autres possibles. Comme on l'a vu, les méthodes agiles*, essentiellement itératives et collaboratives, promeuvent la conception par contrat, ce qui ne veut pas dire que celui-ci doit avoir été finalisé avant que commence le développement, bien au contraire. Celui-ci est bien plutôt le résultat de l'accord progressivement construit entre le maître d'ouvrage et le programmeur concernant les besoins du premier, la situation objective, les contraintes de la machine, etc. De la même manière, Jackson et Zave décrivent le triptyque besoins / spécifications / programme à son état d'arrivée, dans les relations logiques internes qui les caractérisent, et nullement un schéma de déduction linéaire des spécifications à partir des besoins, et du programme à partir des spécifications.

§ 56. L'interprétation contractuelle de la programmation

On ne peut qu'être frappé par le nombre de clauses à garantir afin d'être assuré qu'un programme aussi simple fonctionne, que ces clauses soient du domaine de la situation ou de la machine, c'est-à-dire qu'elles soient du ressort du maître d'ouvrage ou du programmeur.

Il ne s'agit pas là d'un fait accidentel à la programmation. Programmer, c'est insérer une machine dans une situation donnée afin de la modifier : cette insertion est délicate et doit être négociée, des interfaces doivent être construites et des structures initialement indépendantes doivent entrer dans une sorte de synchronicité.

Cette négociation, qu'elle soit formelle ou informelle, souple ou figée, sanctionnée par un contrat juridique ou non, doit aboutir à des engagements réciproques. Programmer c'est, en quelque sorte, garantir à l'aide d'une machine un résultat

dans une situation *si* des conditions s'y trouvent vérifiées.

Nous appelons cette interprétation de la programmation *contractuelle*, afin d'insister sur la réciprocité des engagements entre maître d'ouvrage et programmeur concernant les comportements attendus de la situation et de la machine. Le terme de *contrat* doit ici être pris dans le sens logique que nous avons tenté de décrire, celui d'un schéma bipartite de résolution collaborative de problèmes. Il sous-tend, mais ne requiert aucunement, la relation contractuelle (juridique) qui peut parfois exister entre maître d'ouvrage et programmeur. Ici, rien n'empêche que ceux-ci soient une seule et même personne qui réfléchit à la fois à son problème et à la manière de le programmer, et s'adresse ainsi à lui-même des conditions et des engagements.

Le schéma décrit correspond à une structure de réflexion bien connue des programmeurs. Un livre de référence concernant les bonnes pratiques de la programmation, *The Pragmatic Programmer*, par deux informaticiens à l'origine du mouvement agile*, place la notion de *contrat* au fondement d'une programmation réussie :

> Un contrat définit vos droits et responsabilités, ainsi que ceux de l'autre partie. De plus, il précise les conséquences des cas où l'une ou l'autre partie ne respecte pas le contrat. […] Cette idée est très répandue à travers le monde - formellement et informellement - pour aider les humains à interagir. Pouvons-nous utiliser le même concept pour aider les modules logiciels à interagir ? La réponse est « oui ».
>
> Chaque fonction et méthode d'un système logiciel fait quelque chose. Avant de commencer quelque chose, la fonction peut avoir une certaine attente de l'état du monde [une pré-condition], et elle peut être capable de faire une déclaration sur l'état du monde lorsqu'elle se termine [une post-condition]. […] Le contrat entre une routine et tout agent qui l'appelle peut donc se lire comme suit :
>
> *« Si toutes les pré-conditions de la routine sont remplies par l'appelant, celle-ci devra garantir que toutes les post-conditions et tous les invariants seront vrais lorsqu'elle se terminera. »*
>
> […] Ici, l'accent est mis sur la « programmation paresseuse » (*lazy programming*) : soyez très strict sur ce que vous acceptez avant de commencer, et promettez le moins possible en retour. Rappelez-vous que si votre contrat indique que vous acceptez n'importe quoi et promettez le monde entier en retour, alors vous avez beaucoup de code à écrire[61] !

On retrouve ainsi cette approche au cœur de nombreux concepts informatiques,

61. (D. Thomas et Hunt [1999] 2020, p. 104-107).

comme par exemple l'architecture SOA*[62]. Le langage Eiffel l'intègre dans la construction même des programmes, et l'appelle, significativement *Conception par Contrat* (*Design by Contract*)[63]. Dans ce langage, chaque procédure peut être précédée d'une pré-condition, énonçant les propriétés que doit vérifier l'environnement afin qu'elle puisse être exécutée, et conclue par une post-condition énonçant les propriétés qui seront vérifiées après son exécution. Par exemple, le code 3.3, esquissant une opération de dépôt sur un compte bancaire, énonce qu'on pourra assurer (*ensure*, ligne 4) qu'un dépôt supplémentaire aura été enregistré (ligne 5), et que le solde du compte aura été incrémenté du montant du dépôt (ligne 6), à condition que (*require*, ligne 2) que ce montant ne soit pas négatif (ligne 3).

Code 3.3 – Pré- et post-conditions d'une procédure de dépôt bancaire

```
1 deposit (sum: INTEGER)  -- Add `sum' to account.
2     require
3         non_negative: sum >= 0
4     ensure
5         one_more_deposit: deposit_count = old deposit_count + 1
6         updated: balance = old balance + sum
```

Bertrand Meyer, le concepteur d'Eiffel, affirme que ces structures furent inspirées des méthodes formelles*[64]. En effet, l'idée que la fonction d'un programme est de garantir la réalisation d'une post-condition *si* une pré-condition est garantie par l'environnement d'exécution avant son exécution, prend son origine dans la méthode formelle dite *axiomatique* de Hoare, dont la formule centrale est :

$$A\{\pi\}B \tag{3.6}$$

pour signifier que si la pré-condition A est satisfaite avant l'exécution du programme π, alors la post-condition B le sera à son achèvement[65]. Une autre manière de comprendre cette formule est que le programme π permet de garantir l'effectivité de la *règle « Si A, alors B »*.

C'est cela exactement que prétend garantir le programmeur au maître d'ouvrage et qu'il devrait, idéalement, pouvoir démontrer logiquement : que tels phénomènes surviendront dans la situation si telles conditions initiales y sont réalisées.

62. (Fournier-Morel et al. 2020, p. 67).
63. Voir le chapitre à ce sujet dans la documentation en ligne du langage Eiffel.
64. (Meyer 2008).
65. (Hoare 1969, p. 577).

Un programme d'ordinateur n'est rien d'autre, à ce titre, que *la garantie d'effectivité d'une règle d'inférence* dans une situation donnée. C'est donc au cœur de la définition logique de la programmation que se trouve cette idée d'un contrat passé entre un programmeur et un maître d'ouvrage.

§ 57. Mise à l'épreuve de l'interprétation

L'interprétation contractuelle de la programmation que nous avons exposée permet de rendre compte des cas de figure qui présentaient des difficultés pour l'interprétation formelle.

Le cas le plus simple est celui du caractère imprévisible d'une machine constituée de plusieurs processus concurrents d'exécution du programme[66]. La formule $M, P \vdash S$ indique que le bon fonctionnement de tout programme est soumis à des hypothèses concernant la machine qui l'exécute, autrement dit que tout programme est relatif à la machine. Si l'associativité de l'addition n'est pas une clause de M, alors le programme doit la prendre en charge lui-même. Il y a ainsi des machines qui, à travers leur langage de programmation et leur système d'exploitation, proposent au programmeur des fonctionnalités riches et pré-vérifiées, – en ce cas M énoncera un nombre important de clauses utiles. D'autres machines, au contraire, plus simples, obligent le programmeur à implémenter lui-même la plupart des fonctionnalités dont il a besoin. Un exemple concret de ce choix concerne les librairies* des langages de programmation. Elles fournissent de nombreuses fonctionnalités complémentaires à celles du langage cœur, ce dernier étant en général volontairement limité afin d'en garantir la robustesse et la performance. Le programmeur peut choisir d'utiliser une librairie et d'avoir accès ainsi à des fonctions pré-programmées qui, *pour lu*i, sont partie de la machine. Mais il peut également, pour diverses raisons, choisir de s'en passer et les re-développer dans son propre programme.

Le cas de l'ERP*[67] met bien en évidence le caractère symétrique du contrat entre le maître d'ouvrage et le programmeur. Il oblige cependant à affiner la description des propriétés E de la situation. Dans le texte de 2002 cité plus haut, M. Jackson les présente comme valables inconditionnellement. Cependant dans d'autres articles il reconnaît que, dès qu'elles portent sur le comportement attendu de personnes humaines, ces propriétés ont également un statut optatif (souhai-

66. Voir plus haut, § 52.
67. Voir plus haut, § 51.

table), similaire aux spécifications du programme[68]. Ainsi X. Wang (2016), dans un doctorat qui présente une ontologie formelle du logiciel s'inspirant largement des travaux de M. Jackson, note que les propriétés E, qu'il appelle des *hypothèses* (*assumptions*), peuvent être de nature descriptive ou prescriptive[69]. Cependant à la différence des besoins et des spécifications, le maître d'ouvrage en garde la pleine charge vis-à-vis du programmeur. C'est lui qui doit s'assurer que les personnes humaines coopèrent à la réussite du projet, si bien qu'on doit écrire, en appelant C leur comportement souhaité :

(3.7) $C, E', S \vdash R$

Dans cette formule amendée, l'aspect contractuel du projet est encore plus clairement visible. E' dénote à présent strictement les conditions objectives de la situation qui sont présupposées, et pour la réalisation desquelles aucune action n'est à entreprendre. E' est encadré par deux ensembles de propriétés optatives, C et S doivent être respectivement réalisées par le maître d'ouvrage et par le programmeur afin qu'un troisième ensemble de propriétés R le soit également. On peut noter que C et S ont probablement des formulations en miroir. De la même manière que S décrit les engagements du programme, C décrit ceux des utilisateurs. Il peut s'agir, par exemple, d'un manuel d'utilisation du programme pour ses usagers, et plus généralement de l'ensemble des règles et politiques de l'organisation que le programme supposera appliquées. Comme le note M. Jackson (2002, p. 7), « le problème ne se situe pas à l'interface », c'est-à-dire qu'il ne suffit pas en général de modéliser les interactions entre la machine et la situation pour identifier toutes les conditions requises à la résolution du problème. Les usagers ne doivent pas seulement apprendre à interagir avec la machine, mais également changer certains de leurs comportements non observables directement par elle. Nous avons déjà cité l'exemple de la mise en place d'un ERP* qui oblige les utilisateurs à réaliser leurs achats de produits et services uniquement chez les fournisseurs référencés. Cette obligation n'est pas directement contrôlable par l'ERP, mais au moment de la mise en paiement, le système se bloquera si l'utilisateur n'a pas préalablement édité un bon de commande, ce qu'il n'est possible de faire que si le fournisseur est présent dans la base de données.

Comme nous l'avons déjà dit, un projet de programmation a un caractère biface : afin de résoudre un problème dans une situation donnée, il a en général

68. (M. Jackson 2001a, p. 5).
69. (X. Wang 2016, p. 31).

besoin de la coopération de certaines ressources de cette situation, distinctes de la machine, et pas nécessairement en interaction directe avec elle. Le caractère prescriptif du « contrat » établi par les spécifications porte donc tout autant sur le comportement de la machine que sur celui de ces ressources.

Le cas de l'EDI[70] permet d'observer le caractère symétrique de ce contrat dans sa généralité. Ici, en effet, les ressources dont la coopération est nécessaire pour la réussite du projet sont des machines de part et d'autre. Si par exemple on appelle P_1 et M_1 le programme et la machine du client, et P_2 et M_2 ceux du fournisseur, et que les besoins R communs sont d'échanger les données entre eux sous le bon format, on a :

(3.8) $S_1, E, S_2 \vdash R$

$P_2, M_2 \vdash S_2$

$P_1, M_1 \vdash S_1$

Selon le rapport de forces dans la négociation du contrat S_1 et S_2 seront plus ou moins équilibrés en terme de charge de travail. La difficulté de ces projets reste dans la faible documentation des machines et des programmes initiaux, rendant le projet très empirique par nature.

Il faut noter que rentrent sous ces schémas les problèmes proprement informatiques qui visent à modifier le comportement des machines elles-mêmes. On appelle middleware* les programmes ayant cette finalité, une catégorie un peu plus large que celle de système d'exploitation*. Il s'agit en général d'insérer, dans un écosystème de programmes déjà actifs sur une machine donnée, un nouveau programme venant modifier le comportement de l'ensemble. Il n'y a donc ici qu'une seule machine ($M_1 = M_2$), mais les programmes P_1 (l'écosystème existant) et P_2 (le programme à construire) restent bien distincts. La difficulté est alors de bien distinguer les propriétés invariantes de la machine de l'écosystème de programmes, ce qui se représente en général par des niveaux d'abstractions logicielles, régies par des droits d'accès différents. Lorsqu'on programme à un certain niveau d'abstraction, le niveau sous-jacent ne peut être modifié et apparaît donc comme « la » machine pour le programmeur.

La conclusion de ces remarques est qu'il n'y a nulle polarité homme / machine sous-jacente à l'interprétation contractuelle de la programmation, comme

70. Voir plus haut, § 53.

on pourrait le croire en associant naturellement la situation au monde humain et le programme à la machine. Des machines peuvent se trouver du côté de la situation, c'est-à-dire du problème à résoudre. L'inverse est également vrai : il y a le plus souvent *aussi* des personnes humaines du côté de la machine. Nous avons déjà mentionné le fait qu'un logiciel faisant l'objet d'une transaction commerciale est associé à un contrat de maintenance qui assure le soutien à l'utilisation, les mises à jour techniques, la correction de bogues, etc[71]. Les systèmes informatiques complexes ne tournent jamais tous seuls, et requièrent toujours une certaine part de supervision et de maintenance humaine. On peut évoquer des cas où le travail de la personne humaine est bien spécifié à l'instar du travail de la machine, par exemple lorsqu'il s'agit de « nettoyer » des bases de données afin qu'elles puissent être traitées. Certaines de ces tâches, bien que très routinières, sont difficilement automatisables, notamment lorsqu'il s'agit de corriger des erreurs de saisie, de reconnaître des noms propres, etc. Quant à l'intelligence artificielle, l'enquête de Casilli ([2019] 2021) sur les « ouvriers du clic » montre que les personnes humaines peuvent être essentielles à l'alimentation des algorithmes en données annotées*, c'est-à-dire au fonctionnement de la machine elle-même, et qu'elles doivent être considérées, à ce titre, comme partie intégrale de la chose programmée.

Toutes ces remarques bien sûr, sont importantes pour la question évoquée en introduction, de la *puissance* de l'informatique[72]. Elles peuvent être rapprochées de remarques similaires qui ont été faites concernant les rapports du travailleur à la technologie en général, et sur lesquelles nous reviendrons en conclusion de ce travail.

3.4 Critiques et premières réponses

Dans cette section conclusive, nous commençons par mettre en évidence une propriété épistémique essentielle à la programmation, à savoir que le domaine d'intervention du programmeur, la machine, doit se distinguer de celui du maître d'ouvrage, la situation, par un différentiel supérieur de *certitude*, ou encore de *contrôle*, qu'il doit y garantir (§ 58). Le développement central de cette section § 59 réalise une synthèse de l'interprétation et présente une série d'objections qu'on peut lui

71. Voir plus haut, § 21.
72. Voir plus haut, § 20.

faire, auquel le développement suivant § 60 présente des réponses partielles.

Le dernier développement § 61 récapitule rapidement cette première partie.

§ 58. Le différentiel de contrôle entre machine et situation

La symétrie apparente entre maître d'ouvrage et programmeur dans l'interprétation contractuelle cache une différence essentielle mais implicite. Le contrat ne porte pas seulement sur la répartition des tâches et des responsabilités entre eux. La machine *connecte*, *au sein de la situation*, les propriétés que le maître d'ouvrage peut y réaliser avec celles qu'il souhaite voir advenir. Du point de vue du maître d'ouvrage, la machine est une *garantie de solution* de son problème.

On peut bien sûr objecter que cela n'est pas vrai en pratique, et que l'histoire de l'informatique pourrait autant être racontée par ses « plantages » que par ses innovations. Cependant, ce n'est pas de cela qu'il s'agit, mais de l'*engagement* du programmeur vis-à-vis du maître d'ouvrage, et de la *croyance* de celui-ci en cet engagement. L'informatique, comme profession, s'est construite sur cette promesse de certitude opérationnelle.

Cette garantie est plausible car la machine sur laquelle agit le programmeur est un domaine censé être maîtrisé, dont les configurations peuvent potentiellement être explorées systématiquement. Bien sûr, des pannes matérielles sont possibles, et elles étaient fréquentes dans les premiers ordinateurs. Aujourd'hui encore, lorsqu'elles ne sont pas connectées à un réseau externe, elles restent ouvertes aux aléa de la mise à jour de leurs logiciels, à des incompatibilités non décelées, à des failles de conception. Cependant ces pannes et ces défaillances ont toujours été perçues comme des anomalies à éradiquer, de manière à créer un espace de certitude sur laquelle pourraient se déployer les développements du programmeur. On se rappelle de la vision radicale de Hoare, portant sur la possibilité de droit de vérifier tous les composants d'une machine[73]. Comme nous l'avons vu, c'est exactement ce qui est réalisé pour certains systèmes critiques. La machine est l'espace ou de telles incertitudes peuvent être patiemment mises au pas, ce qu'exprime parfaitement l'image du « rayon cosmique » qui dans son improbabilité, s'oppose à l'« agrément » du maître d'ouvrage, à cet arbitraire subjectif duquel Dijkstra voulait soustraire le programmeur.

Cet arbitraire n'est rien d'autre que celui de la réalité elle-même. Notre portillon de zoo peut avoir été endommagé, ou la ré-initialisation des compteurs peut

73. Voir plus haut, § 48.

avoir été oubliée un matin. La meilleure spécification possible du système d'information d'une entreprise peut être rendue caduque par l'annonce d'une fusion avec une autre organisation, l'évolution de la réglementation peut requérir une re-définition des processus comptables, etc. Les situations évoluent sans cesse, et redéfinissent constamment les problèmes. Il y a donc une dissymétrie fondamentale entre la situation et la machine. La situation, comme nous l'avions dit plus haut[74], est ouverte à l'horizon indéterminé du monde réel, plein d'incertitudes, de nouveauté, de changements, voire de contradictions (entre les personnes), passible d'influences qui, paraissant initialement éloignées, peuvent subitement se révéler d'une actualité et d'une proximité brûlantes. Toute programmation affronte le problème du cadrage[75]. Elle ne peut jamais être certaine d'avoir circonscrit de manière assez large les déterminations de la situation. La machine, au contraire, se présente comme un espace fermé, préparé à l'avance, contrôlé idéalement dans toutes ses composantes.

De ce fait, la représentation graphique proposée par M. Jackson de la situation et de la machine, qu'il place côte à côte (figure 3.10), si elle est particulièrement heureuse d'un point de vue méthodique, car elle met en lumière la distinction aussi bien que l'intersection de leurs domaines de responsabilité, peut être modifiée lorsqu'il faut illustrer leur rapport conceptuel profond. Nous la modifions donc légèrement dans la figure 3.11 pour représenter la machine *à l'intérieur* de la situation. D'abord, la machine n'est pas « hors » de la situation, elle est en son sein, physiquement et conceptuellement, puisque toute la situation doit être amenée à s'adapter à sa réalité. Elle y est certes un espace protégé, dont une partie n'est accessible qu'au programmeur, mais elle y est visible et localisable par ses interfaces, représentées par le cercle intermédiaire. Cet espace est actif dans la situation en ce qu'il en réduit l'incertitude. Il établit des connexions entre certaines propriétés de la situation, de manière à réduire ses configurations possibles. Il est un élément actif ou encore un vecteur de contrôle et de simplification de la situation.

Cependant, lorsque la situation évolue il peut aussi devenir un facteur de rigidité, puisqu'il se trouve partiellement *décalé* par rapport au nouveau besoin. Cet écart doit être compensé par un travail complémentaire au sein de la situation, par exemple un utilisateur va devoir re-formatter certaines données produites par l'ordinateur, voire les corriger. Lorsque cet écart augmente, le programme et la

74. Voir plus haut, § 54.
75. Voir plus haut, § 9.

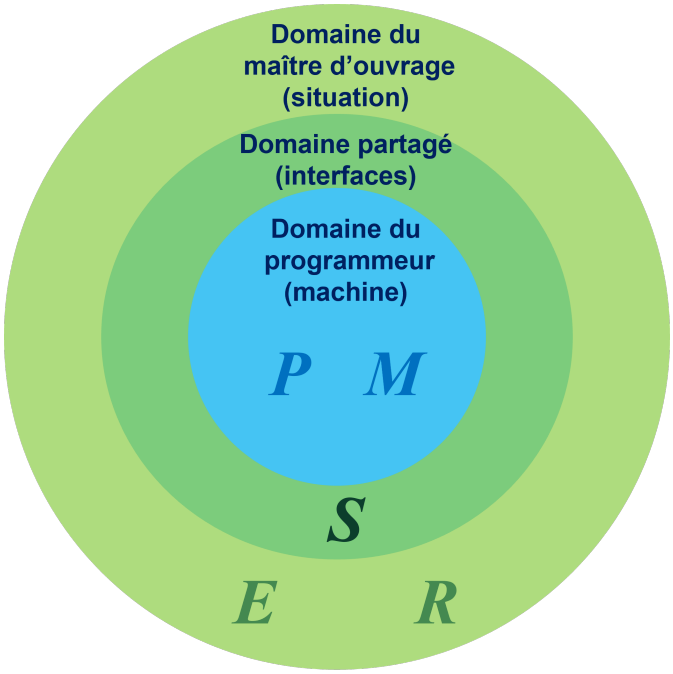


FIGURE 3.11 – La structure de la programmation selon Michael Jackson (modifiée)

machine deviennent des facteurs de complexité et d'incertitude additionnels pour les utilisateurs, dont on ne peut cependant se débarrasser parce qu'il sont toujours critiques pour l'exécution de certaines tâches. Comme on le verra, là se trouve la cause profonde de la *dette informatique*[76].

§ 59. Conclusions et objections

L'interprétation de la programmation qui se dégage ici est toute différente de celle qui en fait une activité formelle, travaillant sur des problèmes mathématiques dans l'indifférence aussi bien aux problèmes réels qu'ils modélisent qu'à la machine qui implémente leur résolution.

Au contraire, la programmation apparaît comme une intervention dans une situation donnée pour y résoudre un problème. Cette intervention se fait par le biais d'une machine en interaction avec la situation, mais également par la mobilisation de ressources dans la situation elle-même qui doivent s'adapter aux besoins propres de la machine. Une part essentielle du travail de programmation consiste ainsi à établir un contrat où deux parties coopèrent pour résoudre un problème

76. Voir plus haut § 30, et plus loin, § 233.

exposé par l'une d'entre elles. Le dispositif créé peut impliquer des personnes humaines et des machines des deux côtés du contrat. En effet, même si la programmation a essentiellement affaire à des machines, le dispositif en jeu dépérit en général rapidement s'il n'est pas animé par des personnes humaines. Ce ne sont donc pas essentiellement les *ressources* apportées par le programmeur au maître d'ouvrage qui doivent être « de nature mécanique », mais la *garantie* qu'il pourra réaliser la situation stipulée à partir de conditions déjà réalisées ou réalisables dans sa situation initiale. Cette garantie « mécanique » est un levier de contrôle de la situation mais également de rigidité si le problème à résoudre évolue. Il est entendu qu'à ce stade de notre réflexion, ce que veut dire ici *mécanique* est encore problématique.

On comprend donc que la polyvalence des ordinateurs pour la résolution de problème ne vient pas d'un langage universel de l'information ou du calcul, mais de l'insertion réussie de machines dans le domaine d'effectivité du problème lui-même. S'il s'agit d'un problème financier, la machine doit être connectée au réseau bancaire ; s'il s'agit d'un problème logistique, elle doit l'être avec le réseau de transports et d'entrepôts pertinents, s'il s'agit d'un problème mécanique, elle doit disposer de capteurs* et d'actuateurs* branchés sur le phénomène à contrôler, etc.

De là suit une seconde remarque : il n'est nullement question ici d'une *traduction* du problème initial en termes algébriques ou en codages binaires ou en circuits électroniques. Les langages de spécification s'expriment souvent en des termes assez proches du langage du problème initial et sont capables d'épouser sa logique interne[77]. Le besoin de précision attendu n'est rien d'autre que celui qu'on demande à un problème dont on cherche une solution systématique. Mais il n'y a pas non plus une grande différence entre des langages de spécification et des langages de programmation, ou en tous cas cette différence parcourt une échelle continue. Certes, l'expression devient un peu plus « rigide » et s'appauvrit d'autant, mais la sémantique de l'ensemble reste fermement dirigée vers le problème qu'il y a à résoudre. Dans l'exemple du portillon de zoo[78], le problème subit des reformulations *horizontales* qui le décalent progressivement du domaine du problème à celui de la machine, et nullement des traductions *verticales* d'un niveau d'abstraction linguistique à un autre.

Depuis cette perspective, les programmes auxquels pensaient les premiers théoriciens de l'informatique, quand ils tentèrent d'appréhender la nature du

77. Voir plus haut § 37. Nous revenons sur ce point en § 200.
78. Voir plus haut, § 55.

phénomène informatique, apparaissent comme des cas très particuliers. Il s'agit essentiellement, dans la discussion des années 1950 à 1970, de programmation fonctionnelle* symbolique, c'est-à-dire de programmes produisant un résultat symbolique (sur un écran, une imprimante, ou une mémoire) à partir de données fournies en entrée (sur un clavier ou via une mémoire). Pendant son exécution, le processus informatique ne peut interagir avec son environnement externe et ne doit pas être interrompu jusqu'à sa terminaison. La polarisation de la réflexion sur ce type de programmes a une explication historique : les premiers problèmes furent posés aux ordinateurs par des scientifiques et des gestionnaires cherchant à calculer des valeurs numériques où à analyser des ensembles de données symboliques. Dans cette histoire des premières applications scientifiques et économiques de l'informatique, on oublie en général un troisième pan, celui des mécanismes servo-moteurs, qui suivit un certain temps une voie parallèle avant de converger avec les deux premiers[79]. Ainsi, le contrôleur de portillon étudié plus haut s'écarte de plusieurs manières du modèle du programme fonctionnel : 1) Il n'a jamais vocation à se terminer. 2) Il est en interaction constante avec une situation. 3) Des informations immédiatement signifiantes sont échangées avec elles, et non des « symboles ». Ces programmes interactifs sont l'objet du chapitre 12. Nous anticipons ce développement en affirmant d'ores et déjà, à la suite de nombreux théoriciens de l'informatique, que ce type de programmes nous semble être le cas général dont il faut partir lorsqu'on réfléchit sur le phénomène informatique.

Dans un programme fonctionnel symbolique, la machine n'*agit* pas sur le problème ultime qu'il s'agit de résoudre. Elle ne contrôle pas, comme dans les systèmes cyber-physiques, des mouvements mécaniques. Elle ne peut pas non plus passer d'ordres ayant une valeur performative dans le monde humain, comme transférer de l'argent, passer des commandes logistiques, etc. Ces actions sont confiées à un agent humain qui prend une décision sur base des résultats symboliques qu'elle produit. C'est cet agent humain qui pose la question à la machine en termes symboliques et interprète son résultat. Il isole par là la machine de la situation, il est ici comme un *interrupteur herméneutique* entre ces deux domaines *aussi réels l'un que l'autre*. Il est significatif qu'ici on dirige la machine à produire des résultats sur des interfaces passives, en général visuelles (écran, imprimante) destinées à la personne humaine. De nombreux cas intermédiaires peuvent ici venir à l'esprit

79. (Bennett 1993).

(par exemple une machine transmettant son résultat à une autre qui, elle, interagit avec la situation, ou encore une machine émettant des rayons lumineux ou des sons guidant des animaux), mais cette continuité ne fait que renforcer notre propos : le programme fonctionnel n'est qu'un cas particulier de programme à interaction pauvre.

De nombreuses objections peuvent être faites à ce renversement de perspective.

1) Un programme interactif peut toujours être décomposé en programmes fonctionnels, et ces derniers sont donc plus élémentaires à ce titre. Aussi n'y a-t-il rien qu'on ne puisse calculer avec les premiers qui ne le soit aussi avec les seconds. L'architecture des interactions entre la situation et le programme est certes une partie importante de la spécification, mais la programmation proprement dite se concentre sur ce que doit faire la machine quand elle est sollicitée par une interaction, et cela peut être décrit fonctionnellement.

2) L'expression d'*informations immédiatement signifiantes*, par laquelle nous avons décrit plus haut les échanges de la machine et de la situation, est trompeuse. Dans le programme du contrôleur du portillon, les fonctions `info_xxx` et `cmd_xxx` sont certes formulées de manière signifiante *pour nous*, mais ne doivent-elles pas être codées en langage binaire afin de devenir effectives au niveau de la machine ? En d'autres termes, ne doit-on pas avouer que notre programme, en faisant appel à ces fonctions sans les expliciter, cache l'essentiel de la programmation, qui a dû être de coder ces interfaces ? Et ne doit-on pas dire que le programme Python3.2 lui-même doit être traduit en instructions machines lors de son exécution ? Ces opérations de traduction d'un problème d'un langage naturel à un langage formel puis à un langage machine ne sont-elles donc pas centrales dans un projet de programmation ?

3) Il n'est pas clair pourquoi le programmeur devrait être responsable de la machine. D'ailleurs, comme nous affirmons que la frontière distinguant la situation de la machine est une affaire d'agrément entre le maître d'ouvrage et le programmeur, il est conceptuellement possible de « passer » la machine entièrement du côté de la *spécification*. Et *de facto*, il y a des cas fréquents où un maître d'ouvrage impose un matériel informatique à un prestataire qui se contente de développer un logiciel qui lui soit adapté. Le programmeur est ici déchargé de toute responsabilité quant au bon fonctionnement de la machine et il est donc engagé dans une pure activité formelle. Cette configuration ne présente-t-elle pas une figure plus pure du programmeur que la précédente, où il semble également encombré du

rôle de fournisseur de matériel informatique ?

4) Si nous insistons sur le fait que le programmeur doit fournir le programme *et* la machine, comme nous affirmons également que la frontière entre machine et programme est également mobile, il est alors possible d'envisager, inversement, des cas où le programmeur se contenterait de fournir uniquement du matériel, et non du logiciel. Nous devons concrètement admettre cette possibilité, puisque lorsqu'il s'agit de systèmes embarqués*, il peut être opportun parfois d'« imprimer » directement le programme sur des circuits électroniques. Le programmeur se contente alors de fournir une machine M pré-programmée pour répondre à la spécification. Mais alors, si cette possibilité est réelle, notre interprétation de la programmation semble pouvoir convenir à la fourniture de tout artefact répondant à un cahier des charges. Elle serait ainsi trop large, décrivant en général tout projet d'ingénierie. L'interprétation formelle distingue au contraire clairement la phase d'implémentation comme traduction du texte du programme en une configuration matérielle. Même si la notion d'implémentation est problématique, comme on l'a vu[80], au moins par là le problème de l'effectivité est clairement isolé, au lieu qu'il semble pouvoir être escamoté dans notre interprétation.

5) Une autre manière d'exprimer ce problème est qu'en réduisant ainsi la programmation à la configuration d'un matériel, on en fait une discipline de technicien ou de bricoleur. On en nie ainsi les dimensions logique et mathématique essentielles qui ont fondé, à tant d'égards, la discipline de la programmation et quasiment toute son armature conceptuelle. De manière plus évidente encore, on voit dans l'exemple du portillon que ce programme doit avoir un fondement mathématique dans l'expression `c > p` (ligne 6 du programme 3.2 ci-dessus) qui exprime que le crédit total cumulé est supérieur au nombre total d'activations du portillon. Sans la modélisation mathématique de ces faits réels par des entiers naturels pouvant être comparés entre eux, le programme ne pourrait être écrit.

6) On peut pousser encore plus loin cette critique. La singularité des ordinateurs parmi toutes les machines, dès Babbage, fut d'être conçues dans la finalité de remplacer la personne humaine dans ses tâches *mentales* et non pas *physiques*. Notre interprétation propose au contraire de les distinguer par la seule modalité de la flexibilité accrue de leur construction et de leur modification. En mettant ainsi en avant une particularité technique de leur production, elle noie par là dans la masse indifférenciée des machines industrielles ce qui fait, justement, leur intérêt philo-

80. Voir plus haut, § 22.

sophique. Ainsi, il existe des programmes qui n'ont pas vocation à effectuer des changements dans le monde, et pour lesquels la notion de *situation* est impertinente. Les résultats produits par des programmes scientifiques servent à l'augmentation du savoir humain, et non au contrôle d'une situation concrète et matérielle. Par exemple, une simulation scientifique peut permettre de simplement valider une hypothèse, et la preuve automatisée d'un théorème mathématique semble être sa propre finalité.

§ 60. Premières réponses

Comme nous l'avons indiqué, nous consacrons le chapitre 12 à la notion de programme interactif. Nous répondrons donc à l'objection *1)* à cet endroit. Il nous suffit ici d'indiquer que le π-calcul*, élaboré par Robin Milner, propose une refondation de la théorie mathématique des processus informatiques à partir de la seule notion d'*interaction*. Il est donc possible de concevoir la notion de programme interactif sans faire appel à celle de programme fonctionnel.

Concernant l'objection *2)*, qui avance que l'essentiel de la programmation se situe dans la traduction progressive de procédures exprimées dans des langages de haut niveau en d'autres exprimant plus directement le fonctionnement de la machine, tout programmeur sait que ce type d'argumentation ressemble à la pelure d'un oignon[81]. Il n'y a pas de niveau d'abstraction qui soit plus « vrai » qu'un autre. Chacun permet de résoudre des problèmes différents, en faisant appel à des fonctions du niveau inférieur. On peut indéfiniment parcourir les niveaux d'abstraction d'un logiciel et décomposer le programme en mouvements de plus en plus précis de la machine : le langage se fera certes plus contraignant, de nombreuses considérations spécifiques à la machine devront être ajoutées, cependant cette décomposition résulte seulement de la factorisation de problèmes complexes en problèmes plus simples et plus communs. Elle est pragmatique et toujours renégociable.

Si tel est le cas, comment alors expliquer que tout projet de programmation implique une ou plusieurs traductions successives d'un langage à un autre ? Selon nous, la seule distinction qu'il est légitime de faire à propos de ces différents stades de réécriture d'un programme – texte en langage de haut niveau, programme compilé, exécutable intégré au système d'exploitation, configurations matérielles

81. Voir l'article déjà cité de (Moor 1978, p. 215-217) à ce sujet.

de la mémoire et du processeur – doit se faire relativement à leur distance opérationnelle vis-à-vis de l'exécution (*run-time*). Cette exécution est toujours un *déclenchement*, c'est-à-dire un événement commandé par un agent de la situation. Il peut s'agir d'une instruction donnée par une ligne de commande ou par le clic d'un bouton, d'un message reçu d'une autre machine, d'une mise en production* complexe, etc. La possibilité immédiate de ce déclenchement seule indique que le programme a atteint sa forme ultime, celle où il se confond avec une simple configuration de la machine. Tant que ce déclenchement n'est pas possible et que des étapes intermédiaires doivent être réalisées afin qu'il le devienne (compilation, intégration, configuration de matériel, etc.) le programme doit encore être considéré comme une description plus ou moins abstraite de la configuration de la machine, en fonction du nombre et de la complexité des transformations encore nécessaires. L'abstraction d'un programme n'est donc qu'une mesure *relative à la machine* sur laquelle il doit être exécuté.

Dans l'objection *3)*, le programmeur est déchargé de toute responsabilité quant au comportement réel de la machine, puisqu'il se contente de fournir au maître d'ouvrage des « textes » que celui-ci a la charge d'implémenter. Il ne fait, en quelque sorte, qu'apporter une expertise au mieux algorithmique, au pire simplement technique de tel ou tel langage de programmation. Ce qu'on décrit là, c'est la figure du *codeur* que nous nous sommes déjà refusés à identifier avec celle, beaucoup plus large, du *programmeur*. Le codeur n'est qu'un poste parmi d'autres dans une équipe de développement. Dans la conception large de la programmation que nous nous sommes donnée[82], celle-ci a pour charge d'assurer l'*effectivité* des programmes et des ordinateurs qu'ils contrôlent dans le monde. Cette conception correspond d'ailleurs à l'image courante que l'on se fait d'un programmeur, comme occupé à modifier le comportement d'une machine, et non pas à faire des calculs logiques.

Il ne s'agit nullement, par là, de faire du programmeur un fournisseur de matériel informatique. Dans le schéma proposé, peu importe qui possède et même qui a choisi le matériel nécessaire à l'exécution du programme. En passant contrat avec le maître d'ouvrage, le programmeur *prend la responsabilité* de la machine, c'est-à-dire qu'il est considéré par les deux parties comme celui qui la connaît le *mieux*, et qui est donc le plus à même d'en comprendre et d'en modifier le fonctionnement. Bien entendu, le programmeur peut avoir de mauvaises surprises lorsqu'il

82. Voir plus haut, § 21.

commence à interagir avec une machine que lui a imposé son client ; des clauses contractuelles peuvent tenter de circonscrire ces déboires possibles : mais il s'agit ici d'un second contrat où le maître d'ouvrage joue un autre rôle, celui de fournisseur de matériel.

Nous ne pouvons répondre aux objections *4)*, *5)* et *6)* à ce stade de notre réflexion. Elles ne sont en effet que des reformulations des questions qui guident ce travail. *4)* pose la question de la différence qu'il y a entre *construire* et *programmer* une machine. Par le premier terme on entend en général une opération qui modifie des états de choses concrets dans monde. La programmation, par opposition, semble n'être qu'un agencement de signes. En gommant cette distinction, notre interprétation semble méconnaître la caractéristique essentielle d'un programme, qu'il s'agit de significations devenant *effectives* dans le fonctionnement d'une machine. Or une de nos questions centrales est celle de l'effectivité des significations qui composent un programme[83].

Nous avons, dès l'introduction[84], présenté les ordinateurs comme des machines incomplètes sans les programmes qui spécifient leur comportement. Cela veut-il dire que l'informatique n'est que la continuation de la mécanique (comme technique de la construction de machines) par d'autres moyens ? Les ordinateurs seraient ainsi seulement des machines très flexibles, pouvant être reconfigurées partiellement sans être démontées et remontées. Mais en insistant ainsi sur cette continuité, notre interprétation semble passer sous silence l'extraordinaire différence qu'il y a entre une simple machine industrielle et un ordinateur, notamment quant à ses capacités de rétroaction sur son propre comportement. De tels problèmes ne peuvent être résolus sans que l'idée même de *machine* soit reconsidérée, ce que nous ferons au chapitre 11.

Cette reconsidération permet de mettre en avant, contre l'objection *5)*, que les machines ne sont pas affaire seulement de bricolage et de technologie, mais également de logique et de mathématiques. Un plan de machine industrielle rend cela assez clair, et on verra que le rapprochement de la logique et de la mécanique est très ancien[85]. Par ailleurs, l'interprétation contractuelle de la programmation donne aux méthodes formelles toute leur place, comme on le voit particulièrement bien dans le cas de systèmes cyber-physiques*, où la leçon de M. Jackson nous

83. Voir plus haut, § 3.
84. Voir plus haut, § 2.
85. Voir plus loin, § 119.

montre que ces méthodes s'appliquent *directement* au problème réel qu'il s'agit de résoudre. Il n'y a pas besoin, pour appréhender la logique interne des programmes, de les traduire d'abord en séquences symboliques ou en codes numériques.

Autrement dit, c'est parce qu'il est possible de décrire par des entiers naturels les deux comptages qui sont d'intérêt dans le problème du portillon (le crédit total reçu et le nombre d'activations) que le contrôle de celui-ci peut être automatisé. Dit plus généralement encore, c'est parce qu'il est possible de décrire logiquement et mathématiquement aussi bien des machines que des situations problématiques du monde réel qu'il est possible d'écrire des programmes qui utilisent les unes pour résoudre les autres. Cette possibilité n'a donc rien à voir avec le fait d'exprimer ou non des programmes en un quelconque langage formel. Elle se rapporte bien plutôt à un vieux problème de philosophie, celui de l'effectivité des mathématiques à décrire le réel. Or ce problème se présente de manière particulièrement aiguë ici, puisque les problèmes dont s'occupe la programmation peuvent être aussi bien théoriques que pratiques. Leur mathématisation ne peut donc pas toujours s'appuyer sur un cadre fourni par les théories scientifiques. Dans le cas des problèmes pratiques de la vie courante, comme celui du portillon, elle semble être formée directement à partir d'une modélisation *ad hoc*, immédiatement opérationnelle, c'est-à-dire qui traite directement de l'effectivité de certaines opérations possibles dans une situation donnée. C'est donc à la question de la nature et de la possibilité de *logiques et de mathématiques des opérations* qu'est suspendue notre compréhension de l'effectivité de la programmation. Nous l'abordons aux chapitres 9 et 10 de ce travail.

La réponse à cette question peut déterminer celle à l'objection *6)*. D'abord, l'opposition entre des tâches « physiques » et « mentales », ou encore entre des problèmes pratiques et des problèmes théoriques « ne changeant rien dans le monde », nous semble se fonder sur une conception assez traditionnelle de la science, pour laquelle les contenus et les questions seraient indépendants en droit des communautés historiques de savants qui les posent et qui les discutent, et sans effet en retour sur ces dernières.

Sans même entrer dans ce débat, ne faut-il pas plutôt retourner le problème, et se demander comment des machines pourraient exécuter des tâches si celles-ci étaient purement « mentales » ? C'est donc à la matérialité et à la force *interne* des signes que nous invite à réfléchir cette objection, une thématique partiellement suggérée par l'interprétation par Jean Lassègue de l'informatique comme d'une

écriture automatique[86]. Si les rouages des machines (ou plutôt leurs circuits électroniques) se prêtent si aisément à des interprétations sémantiques, n'est-ce pas parce que les signes ont déjà le rôle de contrôler et d'orienter nos flux de pensées ?

§ 61. Conclusion de la première partie

Dans cette première partie, nous avons tenté de cerner la nature de la programmation, tout en essayant de construire une base positive d'observations à son sujet, sous la forme d'exemples de programmes, de rappels historiques et d'exposés d'opinions d'informaticiens et de philosophes la concernant. Partant de son caractère protéiforme et de son invisibilisation progressive par son intégration à toutes les technologies et à la plupart des pratiques humaines, nous avons d'abord tenté de la décrire de manière axiologique, en observant les pratiques de l'ingénierie logicielle professionnelle. Le problème de la programmation est de parvenir à une résolution systématique de problème, par une démarche qui ne l'est pas naturellement : notre exemple de programmation amateur l'a assez bien montré. Pour le programmeur professionnel, ce défi est démultiplié par la taille des programmes, leur intégration à des systèmes logiciels toujours plus vastes, et les exigences de robustesse et d'évolutivité sur le long terme. En observant comment la profession a répondu à ce défi, nous avons constaté qu'elle a beaucoup tâtonné historiquement, tout en progressant régulièrement ; il ne semble pas cependant y avoir une formule unique pour parvenir à la systématicité. Deux « philosophies » s'affrontent sur cette question : l'une qui insiste sur le besoin de rigueur et d'industrialisation des pratiques, l'autre qui mise sur l'expérience humaine « artisanale » qui sait s'adapter à la particularité de chaque situation. Les arguments pertinents élevés de part et d'autre semblent pointer vers l'idée que chaque projet de programmation doit trouver son propre équilibre entre ces deux polarités, et qu'il n'y a donc pas, intrinsèquement, une seule « bonne manière » de programmer : la voie de la systématicité est spécifique à chaque situation.

La tentative de donner une définition analytique de la programmation (« un arrangement de signes visant à déterminer le comportement d'une machine, dans une situation donnée, afin de la transformer d'une manière également déterminée ») nous a permis d'explorer les formes les plus variées de programmation ; notamment, nous sommes arrivés à la conclusion qu'il n'était pas possible de donner un critère technique de distinction entre *programmation* et *utilisation* d'une

86. Voir plus haut, § 17.

machine, et que toute utilisation pouvait être vue, à la limite, comme une forme hautement stylisée de programmation.

Ce résultat nous a amené, dans ce chapitre, à caractériser la programmation par son procès épistémique particulier, que nous pensons être correctement capturé par le modèle de Michael Jackson. Celui-ci met en avant un *dialogue* entre un maître d'ouvrage et un programmeur, l'un maître d'une situation qu'il veut faire évoluer, l'autre d'une machine qu'il contrôle parfaitement (dans l'idéal). Ce dialogue doit aboutir à des engagements réciproques qui permettent de cerner la valeur logique d'un programme : il réalise l'engagement du programmeur quant au comportement de la machine dans la situation, si cette dernière est bien telle que le maître d'ouvrage l'a décrite ; et tous deux s'accordent sur le fait que, toutes ces conditions étant réalisées, les besoins du maître d'ouvrage seront satisfaits.

La nature logique du programme est donc, du point de vue du maître d'ouvrage, celle d'une *garantie d'inférence* possible d'une description *S1* de la situation initiale problématique à la description *S2* d'une situation finale satisfaisante. Il est ce qui, *si la situation initiale satisfait S1*, permettra de la transformer de telle sorte qu'elle satisfasse la description *S2*. La machine est, si on peut se permettre ce jeu de mot, configurée par le programme pour être le *moteur de cette inférence* matérielle.

Deuxième partie

La forme générale de la raison pratique ?

Chapitre 4

Concevoir et délibérer (Simon)

§ 62. Introduction

Le chapitre précédent nous dote d'une première compréhension de la structure épistémique de la programmation. Certes, il n'est pas encore possible de répondre à des objections importantes qui peuvent lui être faites, mais celles-ci nous ramènent aux thématiques centrales de notre questionnement. Il s'agit donc à présent de tenter une première approche de la question : *« La programmation est-elle une forme de connaissance ? »*

La programmation nous est apparue comme une activité de résolution de problème à l'aide d'une machine. Pour répondre à notre question, il semble donc nécessaire de caractériser son genre épistémique, la *résolution de problème*, ainsi que sa différence spécifique, le fait de procéder à l'aide de *machines*. Or ces deux termes sont obscurs à ce stade de la réflexion : concernant les machines, nous avons rencontré des difficultés à caractériser la portée qu'il fallait donner à ce terme[1]. Mais l'expression *résolution de problème* a également été prise jusqu'ici dans un sens très large et informel. Elle recouvre aussi bien les problèmes pratiques de la vie quotidienne et des techniques que les problèmes les plus théoriques des sciences et des mathématiques[2] – et c'est pour cela qu'elle se promène si aisément à travers les disciplines, avons-nous remarqué.

Cependant, constituer la *résolution de problèmes* comme objet épistémologique n'est pas un geste évident. Si l'expression est utilisée couramment de nos jours – tout comme *systématiquement* d'ailleurs – il n'est pas anodin d'affirmer

1. Voir plus haut, § 60.
2. Voir plus haut, § 19.

l'*unité* de sens de cette expression à travers des usages si diversifiés. Que peut-il y avoir de commun, par exemple, entre la résolution d'un problème mathématique complexe, qui se déroule dans un cadre théorique déterminé, et le fait de chercher ses clés dans l'appartement ? Rien d'autre, semble-t-il, qu'une lointaine similarité des états psychologiques d'un sujet tendu vers l'accomplissement d'un but. Or, c'est bien vers cette possibilité d'unification que semble pointer la programmation, par son effectivité à résoudre une grande quantité de problèmes, quels que soient leur niveau de complexité (faible ou élevée) et leur domaine (scientifique, technique ou autre) – et par l'idée qu'il serait possible d'avoir des *langages universels* de description et de résolution de problèmes[3].

C'est cette singularité qui a amené Herbert Simon à affirmer, dans son célèbre ouvrage sur les *Sciences de l'Artificiel*, que la programmation était le *prototype* de toute résolution de problème, qu'il fallait envisager comme un acte de *conception* (*design*). L'avènement de l'informatique, donnant enfin son langage propre à cette forme de rationalité, devait permettre à toutes les disciplines pratiques de se constituer enfin en sciences.

C'est à l'examen de cette thèse qu'est consacrée l'essentiel de ce chapitre. Affirmer que la programmation n'est pas *une* forme de connaissance pratique, mais *sa* forme essentielle, nous place d'emblée au cœur de notre problématique, où les points communs, mais également les écarts, voire les contradictions entre la programmation et d'autres formes de savoirs peuvent le mieux apparaître.

Nous exposons la doctrine générale de Simon sur les savoirs pratiques comme conception (section 4.1), puis sur le rôle exemplaire que doit jouer la programmation pour qu'ils puissent se constituer en science (section 4.2). La section 4.3 est consacrée à la discussion critique de ces idées.

Cette discussion nous amène à tenter de comprendre ce que recouvrent les termes de *savoir pratique* et *technique* qui sont, on le rappelle, problématiques à ce stade de notre travail[4]. La dernière section 4.4 de ce chapitre est consacrée à la mise en évidence d'un double sens fondamental en lequel on peut entendre ces termes, grâce à une mise au point terminologique, et à la sollicitation de diverses « philosophies de la pratique » à leur sujet : celles d'Arendt (politique), MacIntyre (éthique) et Bourdieu (sciences sociales).

3. Voir plus haut, § 11.
4. Voir plus haut, § 14.

4.1 Conception et délibération pratique

Herbert Simon, dans son célèbre ouvrage *Les Sciences de l'Artificiel*, dénomme *professions* les activités humaines utiles, constituées autours de corps de savoirs discursifs autonomes et spécialisés nécessaires à leur bon exercice. Parmi les professions on peut compter l'ingénieur, qui a affaire à la production d'artefacts au sens strict, mais également le médecin, le juriste, l'administrateur ou le gérant d'une activité économique, etc. L'intérêt de cette circonscription des pratiques humaines est qu'elle exclue les loisirs, les arts créatifs, les pratiques du corps (notamment les sports) et les pratiques sociales (les rituels, la conversation, etc.) – toutes les pratiques où il est raisonnable de penser que leur réduction procédurale est complexe voire impossible. En même temps, elle est plus large que le simple domaine de la production (représenté par l'ingénierie et l'architecture) puisqu'elle inclut par exemple la médecine et le droit.

L'ambition de Simon, indiquée par le titre de son ouvrage, est de défendre l'autonomie et la particularité de ces savoirs relativement aux sciences académiques, et de montrer qu'il est devenu possible d'envisager leur constitution en disciplines scientifiques à part entière, selon néanmoins une norme de scientificité nouvelle convenant à leurs objets, en tant que ceux-ci ne sont pas *observés* mais *conçus* (*designed*). En d'autres termes, à travers les professions, et grâce au rôle paradigmatique que la programmation est appelée à y jouer, il est possible de mettre en évidence le caractère spécifique de la rationalité technique qui avait échappé aux épistémologues.

Nous allons, dans cette section, exposer la vision générale que promeut Simon des sciences de l'artificiel, avant d'examiner, à la section suivante, le rôle que doit y jouer la programmation. Nous commençons par dégager la notion de *conception* (*design*) que Simon juge centrale dans toutes les réflexions professionnelles (§ 63), avant de la caractériser comme délibération (§ 64) et mettre en évidence son caractère réflexif (§ 65).

§ 63. La conception et l'artefact

L'origine des *Sciences de l'Artificiel*[5] est composite : sa première édition, en 1969, consiste dans la publication de trois conférences données au M.I.T. en 1968,

5. *The Sciences of the Artificial*, (Simon [1991] 1996b), abrégé par la suite en *SA*.

auxquelles il ajoute un article anciennement publié en 1962, *the Architecture of complexity*. En 1981, une seconde édition paraît, où il entrelace les textes des conférences originelles avec ceux de trois autres conférences, données en 1980 à Berkeley. Enfin, en 1996, à l'occasion de la 3ème édition, il intercale avant *the Architecture of complexity* un huitième chapitre visant à tenir compte des nouvelles conceptions de la complexité depuis le lointain article de 1962. Il ne modifie par contre les textes que de manière marginale, en ajoutant principalement des références en notes de bas de page.

Simon aura donc mis à jour *SA* jusqu'à la fin de sa vie (il est mort en 2001). Ce fait atteste sa conviction constante concernant leur validité, mais reflète également le succès croissant de l'ouvrage. Il est en effet l'une des origines principales de la constitution du *design* comme discipline académique, susceptible d'une analyse théorique, et l'expression *science du* design pour désigner cette dernière y trouve son origine[6]. Une convention de traduction doit ici être précisée : nous traduisons *design* par ses équivalents français *conception* et *concevoir*, qui ont l'avantage de pointer vers sa posture réflexive, tandis que *design* insiste plutôt sur son caractère intuitif, diagrammatique. Cependant, nous garderons le terme en italiques pour éviter sa confusion avec les sens plus courants que peut prendre *conception* en français. Enfin, nous garderons l'anglais *design* dès qu'il est question de la discipline académique que Simon entend créer.

C'est dans les trois conférences originelles qu'on perçoit le plus clairement l'argument général de *SA*, dont nous donnons le chapitrage de la dernière édition :

— Chapitre 1 : Comprendre le monde naturel et le monde artificiel
— Chapitre 3 : La psychologie de la pensée : insérer l'artifice dans la nature
— Chapitre 5 : La science du *design* : créer l'artificiel

Le premier chapitre pose le cadre conceptuel de ce que Simon entend par « artificiel », le troisième en explore les fondements épistémologiques, et le cinquième aboutit à un programme prescriptif concernant la discipline académique qu'il appelle de ses vœux. Il perçoit d'emblée cet objectif comme paradoxal, et il faut comprendre les deux premiers chapitres comme visant à le justifier : car en effet, comment admettre que la *conception*, qui est un processus créatif (l'emploi du terme dans le titre du chapitre 5 montre l'insistance de Simon sur ce point) puisse être l'objet d'une étude scientifique ?

La chapitre 5 de *SA* débute par une diatribe passionnée en faveur d'une réno-

6. Voir Buchanan (2009, p. 421-426) et Cross (2006, p. 96).

vation de l'enseignement universitaire des professions[7]. Il ne se satisfait pas en effet de leur réduction à un statut de sciences appliquées, et déplore l'abâtardissement de leur enseignement dans les universités depuis la fin de la Seconde Guerre mondiale : car bien qu'on y trouve toujours des cursus d'ingénierie, d'architecture, de médecine, on n'y enseigne plus vraiment aux étudiants à construire, à concevoir, à guérir. On leur inculque plutôt des savoirs analytiques qui, prétendument, leur seront utiles dans l'exercice de leur métier, comme par exemple les mathématiques appliquées pour le mécanicien, la science des matériaux pour l'architecte, la biologie pour le médecin, etc. D'après lui, ce déplacement a une cause institutionnelle : l'uniformisation, après la Seconde Guerre mondiale, des cursus universitaires et des diplômes d'études supérieures, exigeant des disciplines passibles d'évaluation objective, de thèses de doctorat, d'articles de recherche, etc.

> Il y avait [à ce phénomène une cause] très évidente. Les écoles professionnelles […] aspiraient à la respectabilité académique. Selon les normes en vigueur, la respectabilité académique requiert un sujet intellectuellement difficile, analytique, formalisable et enseignable. Or dans le passé, une grande partie, sinon presque tout de ce que nous savions sur le *design* et les sciences artificielles était intellectuellement léger [soft], intuitif, informel et avait l'air de recettes [*cook-booky*]. [8]

Or si Simon pense que l'expertise technique ne peut être dérivée mécaniquement des sciences, il pense également qu'elle vaut mieux que l'expérience :

> Nous n'avons nul besoin de revenir aux recettes de cuisine qui ont originellement causé le discrédit de la *conception* et qui l'ont chassé du programme de formation de l'ingénieur. Car il existe aujourd'hui un nombre considérable d'exemples de processus réels de *conception*, de types très différents, qui ont été entièrement définis et ont été, pour ainsi dire, fondus dans le métal, sous la forme de programmes informatiques[9].

Qu'entend donc Simon par ce terme *conception*, et pourquoi pense-t-il légitime de le mettre au fondement de champs disciplinaires aussi variés que l'ingénierie, le droit ou les affaires ? Il semble en effet avoir trait à la production d'artefacts, et donc à la technique plutôt qu'au domaine pratique en général. Et pourquoi la programmation, que nous avons plutôt rapprochée des logiques de l'action, serait-elle appelée à jouer un rôle si central dans sa refondation rationnelle ?

La *conception* porte sur les artefacts, mais un artefact n'est pas d'abord un

7. *SA*, p. xii-xiii.
8. *SA*, p. 111-112.
9. *SA*, p. 134.

objet matériel ; il est défini fonctionnellement, c'est-à-dire par l'usage qu'on veut en faire dans un environnement donné. Cela veut dire qu'il n'est pas d'abord défini par ses éléments constitutifs, ses matériaux par exemple. Ceux-ci sont plutôt des contraintes dont la technique doit tirer parti, ou qu'elle doit lever en leur substituant d'autres éléments. L'objet propre de la *conception* est plutôt de trouver des solutions pour les finalités recherchées en tenant compte des contraintes posées. Ces solutions prennent la forme d'interfaces entre un système interne, doté de lois de comportement déterminées, et un environnement externe, qui pose des contraintes en même temps qu'il évalue l'adéquation du système aux objectifs fixés. On reconnaît ici une esquisse du schéma de la machine et de la situation exposé au chapitre précédent, mais Simon n'a pas à l'esprit uniquement les systèmes informatiques. Il prend l'exemple de l'invention anglaise des montres marines, qui permit d'éviter le dérèglement par le roulis de la mer des mécanismes horlogers. Une « bonne » montre est une montre adaptée à ses conditions spécifiques d'utilisation.

> Un artefact peut être conçu comme un point de rencontre – une *interface* dans les termes d'aujourd'hui – entre un environnement *interne*, la substance et l'organisation de l'artefact lui-même, et un environnement *externe*, les alentours au sein desquels il opère. Si l'environnement interne est approprié à l'environnement externe, ou vice versa, l'artefact servira son intention[10].

Cela veut dire que l'objet propre de toute technique est de concevoir *des interfaces* :

> Dans le meilleur des mondes possibles – au moins pour un concepteur – nous pourrions même espérer […] pouvoir caractériser les propriétés principales d'un système et de son comportement sans décrire en détail ses environnements externe et interne. Nous pourrions viser une science de l'Artificiel qui trouverait dans la simplicité relative de l'interface sa source primaire d'abstraction et de généralité […] La description d'un artifice en termes d'organisation et de fonctionnement – son interface entre ses environnements externe et interne – est un objectif majeur de l'invention et de l'activité de *conception* [11].

Cette conception de la technique comme un travail sur les interfaces entre deux environnements, interne et externe, est l'intuition majeure de Herbert Simon lorsqu'il rédige les *Sciences de l'Artificiel*. Le technicien ne travaille jamais que sur des systèmes naturels (matériaux, corps organiques, artefacts utilisés à leur tour comme sous-systèmes). Ce qui est proprement artificiel dans l'artefact est l'agen-

10. *SA*, p. 3.
11. *SA*, p. 9.

cement de ses parties pour accomplir une fonction dans son environnement.

> Nous devons nous garder d'équivaloir *biologique* et *naturel*. Une forêt est peut-être un phénomène de la nature ; une ferme ne l'est certainement pas. […] Un champ labouré n'est pas plus partie de la nature qu'une rue asphaltée. […] Ces choses que nous appelons des artefacts ne sont pas séparés de la nature. Ils ne sont aucunement autorisés d'ignorer ou de violer les lois de la nature. Et en même temps ils sont adaptés aux buts humains et à leurs finalités[12].

L'artificiel n'est pas le « non-naturel ». En particulier, il faut cesser d'assimiler l'opposition entre le naturel et artificiel à celle qu'il y aurait entre le vivant et la machine, car tous deux suivent les mêmes lois naturelles. La *conception* ne se réduit pas cependant à *appliquer* les résultats des sciences – connaissance des lois physiques, des matériaux, des taxonomies de phénomènes. Elle peut mobiliser tout cela, mais ne s'en laisse pas déduire, car l'agencement de ces éléments pour parvenir aux fins désirées est son objet propre. Il y a ici un écho de la vieille association grecque de la *technè* avec la *mètis*, la ruse qui se joue de la nature pour parvenir à ses fins[13]. Une interface en effet est un agencement de milieux soumis à des lois naturelles distinctes afin de les faire coopérer pour faire advenir une situation désirée. Plus fondamentalement encore, Simon retrouve la définition qu'Aristote donnait de la rationalité délibérative comme occupée des choses contingentes, par opposition à la rationalité théorique occupée des choses nécessaires[14] :

> L'ingénierie, la médecine, les affaires, l'architecture et la peinture ne sont pas concernés par le nécessaire mais par le contingent – non par les choses telles qu'elles sont, mais telles qu'elles pourraient être – en bref par la *conception* [15].

Le paradoxe de la position de Simon est tout entier ici : il pense que la *conception*, qui est donc pour lui un autre nom de l'activité rationnelle délibérative en général, a ses principes et ses lois propres, et qu'elle peut donc être l'objet d'une science, contrairement à Aristote.

§ 64. Conception et délibération

Il est utile de s'arrêter sur cette assimilation de la *conception* et de la délibération. À première vue, elle ne va pas du tout de soi. La *conception*, dans le sens

12. *SA*, p. 3.
13. (Warin 2022).
14. *Éth. Nic.* VI.1. Nous étudions la position d'Aristote au chapitre 6.
15. *SA*, p. xii.

courant, porte sur des objets matériels. Elle se visualise par des plans et des esquisses de l'artefact à produire. La délibération, au contraire, porte sur l'action, sur un cours d'événements qu'on veut voir advenir pour parvenir à ses fins. Elle se visualise par des diagrammes de séquence qui représentent les actions.

Pour Simon, cette distinction est superficielle, car ce qu'il vise par *conception* est bien la rationalité délibérative en général :

> Les ingénieurs ne sont pas seuls concepteurs professionnels. Conçoit quiconque élabore des plans d'action visant à changer des situations existantes en d'autres qui leur sont préférées. L'activité intellectuelle qui produit des artefacts matériels n'est pas fondamentalement différente de celle qui prescrit des remèdes à un patient, ou qui élabore un plan commercial pour une société ou une politique de couverture sociale pour un État. La *conception*, ainsi construite, est au cœur de tout apprentissage professionnel ; elle est la marque principale qui distingue les professions des sciences[16].

On peut observer précisément Simon s'approprier le terme *design* dans deux textes formant dyptique, *the Logic of Rational Decision* (1965) et *the Logic of Heuristic Decision-Making* (1967) précédant donc de peu les conférences au M.I.T., qui s'y réfèrent [17]. Le terme apparaît déjà dans les écrits antérieurs de Simon, dans son sens traditionnel lié à l'ingénierie. Il note déjà le parallèle entre élaboration de plan d'action et *design* d'ingénierie[18], mais c'est dans notre premier article de 1965[19] qu'il commence à l'employer comme synonyme, et dans l'article de 1967 qu'on trouve sa première thématisation comme concept central de la rationalité pratique.

L'objet des deux articles est la « théorie de l'action », et Simon se targue de fonder ses idées sur l'observation de cas concrets de prise de décision par l'homme d'affaires, l'ingénieur, le médecin, le militaire, etc. Il distingue deux phases dans la prise de décision : une phase concernant la fixation du problème à résoudre – par exemple un choix parmi plusieurs priorités concurrentes, et une phase concernant la délibération stricte, c'est-à-dire la détermination des moyens pour résoudre le problème. Celle-ci est introduite dans les termes suivants :

> Décisions de *conception*
>
> La seconde étape dans la prise de décision est d'élaborer ou de découvrir de possibles lignes d'action. C'est l'activité qui dans des champs comme l'ingénierie et

16. *SA*, p. 111.
17. Textes repris dans (Simon [1967] 1977, chp. 3.1 et 3.2).
18. (Simon [1947] 1997, p. 312-33).
19. (Simon [1967] 1977, p. 148-149).

> l'architecture est appelée *design*, dans le champ militaire *planification* ; en chimie *synthèse* ; en d'autres contextes *invention* ; *composition* ou ce terme suprêmement approbateur *création*. Dans les tâches routinières et répétitives de prise de décision, la *conception* joue un rôle mineur, car les options possibles pour l'action sont déjà disponibles sur étagère et mobilisables dès que l'occasion le requiert. Dans des situations légèrement moins structurées et répétitives, la *conception* pourrait ne nécessiter qu'un assemblage relativement sans problème d'actions possibles tirées de composants préfabriqués. Dans la plupart des affaires humaines cependant, l'étape de *conception* dans la prise de décision occupe une bien plus large part de la capacité de traitement de l'esprit que celles de direction de l'attention ou du choix.
>
> La *conception* a pour objet l'élaboration de moyens en vue de fins[20].

Le schéma de la *fin et des moyens* est donc bien le point de départ de Simon concernant la rationalité pratique. Ce schéma remonte à Aristote, qui l'assimile en *Éth. Nic.* III.3 au processus même de la délibération : lorsqu'on cherche à atteindre une fin donnée, on examine les moyens qui permettent de la réaliser, et chaque moyen nécessaire devient à son tour une nouvelle fin dont il s'agit d'explorer la faisabilité par la même méthode[21]. Simon a longtemps pensé que ce schéma était la forme générale de la rationalité pratique, et c'est sur cette base qu'il construit, avec Shaw et Newell, l'un des premiers programmes emblématiques de l'intelligence artificielle, le General Problem Solver (GPS, en français : *solveur général de problèmes*)[22]. Le programme a une structure inductive et utilise une fonction d'évaluation des états du système en rapport avec l'état final recherché pour tester chaque action possible et choisir celle qui l'en rapproche le plus[23].

Simon a conscience d'une certaine incongruité du sens nouveau qu'il souhaite donner au terme *conception*, puisqu'on ne se représente pas l'ingénieur comme planifiant une ligne d'action ; il lui faut donc expliquer cette unité entre l'action productive et l'action en général :

> On pourrait objecter que notre sens proposé pour *conception* exclut presque tout ce qui est dénommé ainsi en ingénierie et en architecture. La solution typique d'un problème d'ingénierie ou de la *conception* architecturale n'est pas un programme dans l'espace des actions, mais une description (ou image) d'un objet dans l'espace des états - un plan ou une élévation, par exemple. Mais la raison pour laquelle cette

20. (ibid., p. 160).
21. Voir plus loin, § 98.
22. (Newell, Shaw et Simon 1959).
23. (Simon [1967] 1977, p. 167). Nous donnons plus de détail sur ce programme important, auquel nous nous référons souvent, en annexe A.

> description est considérée comme une solution du problème de la *conception* est que les étapes permettant de terminer la traduction de cette description en un programme d'actions (un programme de fabrication ou de construction) ne sont pas problématiques. La seule action que l'ingénieur ou l'architecte doit effectuer (si je puis me permettre une simplification un peu héroïque) pour réaliser la *conception* est de la confier à un département de fabrication ou à un entrepreneur en construction[24].

La distinction entre objet matériel et plan d'action est ici remplacée par une distinction entre espace des actions et espace des états, une formulation inspirée de la physique mathématique où on parle souvent de matrices de transitions sur des espaces d'état. Le cas de l'ingénierie est donc spécifique en ce qu'elle dispose déjà de procédures de production qui savent, à partir d'un plan donné, réaliser l'objet correspondant, de sorte que l'ingénieur peut se focaliser sur la *conception* du plan seulement. Mais dans le cas général, l'objet de la réflexion pratique est de mettre en relation des actions possibles avec des situations désirées.

Cela explique aussi pourquoi, réciproquement, des délibérations qui semblent porter sur des actions à entreprendre sont aussi de la *conception*. Un médecin, en apparence, prescrit des actions à mener : prendre tels médicaments à telle fréquence, faire tels ou tels exercices, etc. Mais ce traitement a pour objectif de modifier la situation actuelle (la maladie) en une situation désirée (la santé) en modifiant certaines de ses caractéristiques (tel équilibre hormonal, telle tonicité musculaire, etc.) À l'opposé de l'ingénieur qui décrit directement l'état final désiré et laisse implicite les actions à mettre en œuvre pour ce faire, le médecin ne décrit que celles-ci et laisse implicite les situations intermédiaires et finale qu'il vise en réalité. Le général d'armée et le joueur d'échecs sont dans la même configuration que le médecin : ils n'explicitent que les actions à mener ou les coups à jouer, mais ce qu'ils ont bien à l'esprit, ce sont les situations qu'ils aimeraient voir advenir sur le champ de bataille ou sur l'échiquier.

Dans le cas général de la délibération, rien n'est cependant évident – ni les situations à faire advenir, ni les actions à mener, et il faut concevoir les possibilités des deux côtés :

> Dans ces situations du monde réel, complexes, nous pouvons nous représenter le processus de *conception* comme visant à tisser un réseau serré de connexions entre les actions physiques réelles (les actions des tourneurs et des charpentiers) d'une part, et les états du monde réalisés (moteurs électriques, immeubles de bureaux)

24. (Simon [1967] 1977, p. 164).

> d'autre part. De nombreuses parties de ce réseau peuvent être tissées en utilisant des motifs traditionnels, sans nouvelle résolution de problème que celle du choix parmi des options connues. À d'autres points du réseau, cependant, tisser ensemble le tissu des moyens, venant de la gauche, avec le tissu des fins, venant de la droite, nécessite l'exploration de nouvelles options. C'est à ces interfaces (il peut y en avoir plusieurs) que se déroule l'activité de *conception*[25].

On voit dans ce texte Simon substituer très précisément et même très visuellement, par cette métaphore des tissus, le schéma de l'interface à celui de la fin et des moyens. Outre la projection des actions dans un espace prédéfini, Simon opère un second déplacement par rapport au schéma aristotélicien de la délibération, en substituant à la séquence linéaire de l'analyse (« s'il n'y en a qu'un seul pour arriver à cette fin, on cherche comment il permet d'y arriver et aussi par quel moyen ce moyen lui-même peut être atteint... »[26]) la figure du réseau qu'on doit connecter en plusieurs points. Le schéma aristotélicien pouvait à la rigueur prendre la figure d'un arbre, car le texte d'*Éth. Nic.* III.3 peut laisser entendre que plusieurs moyens sont nécessaires pour atteindre une fin, et qu'il faut donc appliquer la même analyse à chacun d'entre eux. Mais dans le schéma simonien du réseau il y a une symétrie complète de la disposition des fins et des moyens. Il est en effet illusoire de penser qu'on cherche une chose unique dans la délibération ; cette illusion est sans doute liée à la focalisation de l'attention sur la propriété saillante de l'état du monde qu'on cherche à réaliser ; mais il y a toujours des contraintes, souvent implicites, qu'on impose à toute situation cible, par exemple celles de ne pas entamer les ressources dont nous disposons au-delà d'un certain seuil. Ces ressources ne sont pas des « moyens » mais également des propriétés de l'espace des états. Les « moyens » sont, à strictement parler, les actions que nous pouvons lancer. Celles-ci peuvent parfois être décrites par des plans ou des instructions écrites, dont les routines de réalisation sont alors connues – ce que signifie *un réseau serré de connexions*.

Cette expansion du schéma linéaire ou arborescent de la fin et des moyens dans un schéma de maillage des actions et des états permet de reformuler la notion de problème ; il ne s'agit plus de trouver le chemin des moyens à la fin, mais de résoudre une équation systémique, où plusieurs contraintes doivent être satisfaites dans l'espace des possibles. Le chemin, si on peut encore l'appeler ainsi, est un optimum dans un espace de chemins possibles, qui résout au mieux un système de

25. (ibid., p. 164-165).
26. Ce texte est cité et commenté plus loin, § 98.

contraintes ; par là Simon ramène le schéma traditionnel de la fin et des moyens à un schéma plus général, celui de la recherche de solutions dans un espace complexe.

Un espace de problème n'est ainsi rien d'autre qu'un ensemble d'interfaces possibles entre un agent capable d'actions et une situation caractérisée par des états. Concevoir une interface revient à trouver les règles qui ajustent systématiquement les actions de l'agent à l'état de la situation. Ce schéma général permet de voir l'identité profonde de la *conception* et de la délibération, ainsi que de la production et de l'action. Concevoir un artefact n'est qu'une certaine manière d'envisager la reconfiguration d'une situation matérielle en vue d'une fin, ce qui n'est rien d'autre qu'un plan d'action.

§ 65. La réflexivité de la *conception*

Si Simon garde *conception* plutôt que *délibération* pour désigner l'activité de la raison pratique, c'est qu'il va employer ce terme pour désigner la délibération au second degré qui consiste à identifier les bonnes stratégies de recherche dans l'espace de problème et à simplifier cet espace lui-même. Ce développement vise à exposer cette *réflexivité* possible de la *conception*, qui va lui permettre de s'établir comme science.

La caractérisation de la *conception* comme « stratégie de recherche » provient de la *recherche opérationnelle*, c'est-à-dire l'étude des méthodes mathématiques appliquées aux problèmes de la décision, comme l'allocation de ressources, qui se développèrent pendant la seconde guerre mondiale, où elles furent un outil critique de planification. Simon les intègre à son programme d'enseignement dès 1949[27] et rappellera souvent leur consanguinité avec l'intelligence artificielle[28]. Bien que la recherche opérationnelle ne soit qu'une forme très spécifique de recherche, qui exige une représentation du problème par un système d'équations sur des grandeurs continues[29], Simon en retient d'abord que les problèmes pratiques peuvent être objets de calcul :

> Ce qui était vraiment nouveau pour l'économie dans la recherche opérationnelle, c'est cet intérêt pour la rationalité procédurale – pour trouver des procédures efficientes calculant des solutions réelles à des problèmes de décision concrets. Les

27. *SA*, p. 139.
28. (Simon 1987).
29. *SA*, p. 28.

> exigences de calculabilité conduisirent à deux sortes de déviations par rapport à l'optimisation classique : la simplification du modèle pour rendre possible le calcul d'un optimum, ou sinon, la recherche de choix satisfaisants plutôt qu'optimaux[30].

Délibérer sur un problème simple, en effet – par exemple, comment se rendre de l'Étoile à Nation par les transports en commun – n'est pas ce qui intéresse vraiment la science du *design* que veut mettre au jour Simon. Les problèmes complexes ont des espaces de possibilités d'une grandeur telle qu'elle voue très vite à l'échec une recherche aléatoire. Ce qui est requis – et qui est vraiment l'objet de la *conception* selon Simon – c'est d'élaborer des stratégies de recherche qui permettent de trouver le plus efficacement possible des solutions acceptables dans l'espace de problème. Cet espace est lui-même composé, rappelons-le, des séquences autorisées d'états et d'actions. La science du *design* porte donc sur une recherche au second degré.

Cet aspect réflexif est présenté clairement dans l'article déjà cité de 1965 concernant la théorie de l'action. Cet article distingue trois niveaux d'impératifs guidant toute délibération :

> Dans l'activité de résolution de problèmes, nous devons considérer les relations entre trois ensembles d'impératifs : (A) les impératifs prescrivant les objectifs de l'action [*performance*] et ses contraintes ; (B) les impératifs prescrivant des programmes d'action [*performance programmes*] pour atteindre ces objectifs ; (C) les impératifs prescrivant des programmes de résolution de problème [*problem-solving programmes*] pour découvrir et évaluer les programmes (B)[31].

Par cette distinction, Simon veut pointer qu'en général il n'y a pas déduction de (A) à (B), c'est-à-dire des objectifs aux moyens. La plupart du temps, les programmes (B) sont proposés sur base de l'expérience passée, de programmes déjà connus dont on suppose, sans le démontrer, qu'on peut les transposer au cas présent. Il existe des cas où on dispose d'une méthode (C) pour déduire exactement (B) à partir de (A), comme dans certains problèmes de recherche opérationnelle ; mais dans le cas général ces méthodes (C) ne garantissent pas l'optimalité de la solution, ni même qu'elles en trouvent toujours une ; elles sont *heuristiques* ; et c'est dans la *conception* et l'amélioration de ces méthodes que se situe proprement le lieu de la réflexion pratique – le lieu de l'invention.

Dans un espace de problèmes ayant une très large combinatoire, certaines heuristiques (C) permettent d'améliorer l'efficacité de la recherche en éliminant des possibilités. Par exemple, s'il s'agit de résoudre un système d'équations portant

30. (Simon 1976b, p. 76).
31. (Simon [1967] 1977, p. 149).

sur 10 inconnues représentant les chiffres 0..9, la taille de l'espace est de 10! si on teste toutes les combinaisons possibles. Parvenir à fixer la valeur d'une seule grâce à une observation sur les contraintes réduit la taille d'un facteur 10, puis 9, etc.[32]

Les heuristiques ne portent donc pas seulement sur la stratégie de recherche, mais également sur la simplification de l'espace de problème. C'est ici qu'on voit mieux cette dimension réflexive de la *conception*. Elle n'est pas contrainte par l'espace de problème, car au contraire son objet est justement de le représenter correctement. Simon va jusqu'à dire que, d'une certaine manière, « résoudre un problème veut dire le représenter de manière à rendre sa solution transparente[33] ».

Une stratégie classique est de parvenir à découper l'espace de problèmes en sous-espaces indépendants ou quasi-indépendants[34], chacun ayant une taille raisonnable pour une recherche classique. Par exemple, si je cherche un restaurant parmi les 12000 existant à Paris, il n'est pas efficace de les examiner un par un. En répondant d'abord à deux questions ayant chacune environ vingt réponses possibles, comme *« dans quel quartier veux-je dîner ? »*, *« de quel type de cuisine ai-je envie ? »*, je réduis mon problème à une trentaine possibilités (en moyenne). J'ai décomposé un problème large en trois problèmes simples à résoudre.

Il est donc fréquent pour la résolution de problèmes complexes de procéder par abstractions hiérarchiques, qui simplifient les espaces d'action et d'état de manière à focaliser la recherche sur quelques grandes options, en laissant de côté tout le détail ; par exemple, pour l'ingénieur qui construit un chemin de fer à travers un pays montagneux, il s'agit de choisir entre trois ou quatre grandes options, et seulement ensuite d'établir des plans plus détaillés sur base de calculs précis : de multiples *espaces de planification* peuvent être ainsi distingués au cours d'une démarche de *conception*. L'exemple de la *conception* d'une route est un des exemples préférés de Simon, sans doute à cause de son caractère naturellement métaphorique de toute élaboration d'une ligne d'action[35]. Il n'y a donc pas de différence de nature entre méthodes et procédures ; les méthodes sont des esquisses de procédures, des indications de balises qui seront précisées en procédures lors de leur application à des situations concrètes. Leur indétermination n'est que la condition de leur adaptabilité.

Un plan général se détaille ainsi progressivement ; chaque décision supérieure

32. *SA*, p. 54-57.
33. *SA*, p. 132.
34. Sur cette notion de *quasi-indépendance*, voir *SA*, p. 197.
35. Voir par exemple (Simon [1947] 1997, p. 109), (Simon 1972, p. 172), *SA*, p. 125.

cadre les problèmes posés aux niveaux inférieurs et les décisions qu'il faudra y prendre, et en retour chaque chemin complexe apparaît comme la composition d'une hiérarchie plus élémentaire. Cette vision est quelque peu schématique, car Simon sait bien qu'il est rare de pouvoir décomposer nettement un problème en sous-problèmes indépendants ; mais il pense qu'il est possible, dans la plupart des cas, d'aboutir à des représentations *quasi-modulaires* de l'espace de problème, où les sous-espaces ont des interactions *faibles* entre eux, ce qui permet de les étudier de manière séparée, *au premier ordre*, et d'étudier leur composition comme un problème séparé *d'intégration*, qui est justement celui du niveau supérieur. On voit ici toute la pertinence de la notion de *conception*, qui doit parvenir à une telle représentation efficace de l'espace de problème, identifier ses lignes de découpe qui permettront de simplifier radicalement son traitement. Savoir résoudre un problème n'est rien d'autre qu'en avoir la bonne représentation. Ainsi, chaque type de problème développe son propre langage pour les représenter :

> Nous savons que les problèmes peuvent être décrits verbalement, en langage naturel. Ils peuvent souvent être décrits mathématiquement, en utilisant les formalismes standard de l'algèbre, de la géométrie, de la théorie des ensembles, de l'analyse ou de la topologie. Si les problèmes concernent des objets physiques, ils (ou leurs solutions) peuvent être représentés par des plans, des dessins techniques, des rendus ou des modèles tridimensionnels. Les problèmes liés aux actions peuvent être attaqués avec des diagrammes de flux et des programmes[36].

Simon ne se satisfait cependant pas de l'état actuel de ce qui devrait être une *théorie des représentations* :

> Une première étape vers la compréhension de n'importe quel ensemble de phénomènes est d'apprendre quels types de choses on y trouve – développer une taxonomie. Cette étape n'a pas encore été franchie en ce qui concerne les représentations. Nous n'avons qu'une connaissance sommaire et incomplète des différentes manières dont les problèmes peuvent être représentés et encore moins de l'importance de ces différences[37].

Ceci se voit en ce que parfois la stratégie standard, c'est-à-dire la recherche d'une découpe modulaire d'un espace, n'aboutit pas. Il faut alors *étendre* la représentation de l'espace de problème. Un exemple classique sur lequel revient souvent Simon est celui de « l'échiquier mutilé », duquel on a retiré deux cases à deux extrémités opposées diagonalement (voir figure 4.1). La question est alors de savoir si

36. *SA*, p. 133.
37. *SA*, p. 133.

on peut paver les 62 cases restantes de l'échiquier avec 31 dominos. Une recherche empirique pour tenter de réaliser un tel pavage ne mène nulle part. Il faut opérer un changement de représentations pour trouver la solution, qui semble alors évidente. Si on prête attention à l'alternance de couleurs des cases de l'échiquier, on s'aperçoit que chaque domino doit nécessairement couvrir deux cases de couleur opposée. Or l'échiquier mutilé contient 32 cases d'une couleur, et 30 cases de l'autre. Cela prouve qu'il n'y a pas de solution du problème. Cette méthode implique de *voir* la couleur des cases, non plus comme une propriété accidentelle, mais essentielle de l'échiquier. En introduisant un nouveau concept, la parité

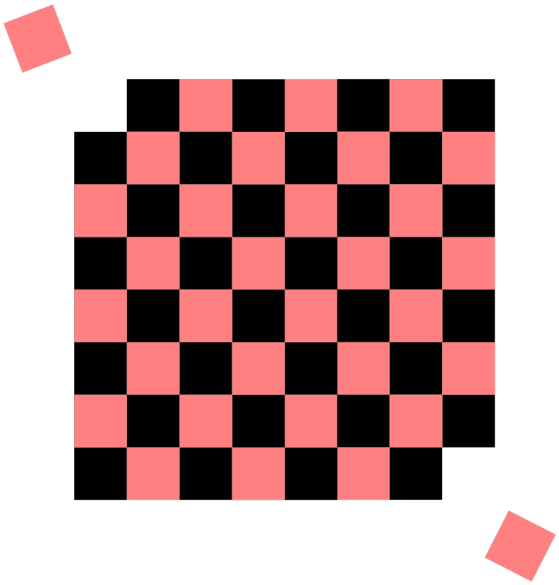


FIGURE 4.1 – Le problème de l'échiquier mutilé

des cases, c'est tout l'espace de problème qui s'en trouve radicalement simplifié. Cette opération se déroule dans ce que KAPLAN et SIMON (1990, p. 380) décrivent comme le « méta-espace des espaces de problèmes possibles pour le problème de l'échiquier mutilé ».

La *conception* a donc une dimension essentiellement réflexive. Elle devient objet d'une étude scientifique non par la résolution de problèmes concrets, mais par l'étude des stratégies de résolution de problèmes, soit qu'il s'agisse d'optimiser la recherche dans un espace de problème donné, soit de reconfigurer cet espace

lui-même pour mieux le représenter. Cet aspect réflexif montre également sa dimension créative, qu'évoque naturellement le terme *conception*.

> La théorie classique de la décision s'est occupée de choix parmi des options déterminées, alors que la *conception* concerne la découverte et l'élaboration d'options. […] Les évaluations et les comparaisons qui prennent place lors de ce processus de *conception* ne sont pas, en général, des comparaisons entre des *conceptions* complètes. Les évaluations ont lieu, d'abord, afin de guider la recherche – l'élaboration de la *conception* elle-même. Elles fournissent la base pour décider si la *conception* doit être élaborée dans une direction plutôt qu'une autre[38].

L'activité de *conception* est recherche et invention de la bonne heuristique et du bon espace de problème, ce qui se produit par un processus créatif dont chaque étape créée de nouveaux problèmes, qui ouvre de nouvelles possibilités. Dans les problèmes « mal structurés », c'est-à-dire dont on n'a au départ qu'une spécification vague ou incomplète, le processus de *conception* consiste à sans cesse modifier, partitionner et restructurer l'espace du problème global jusqu'à ce que celui-ci se trouve bien posé. Simon étudie ici le processus créatif de l'architecte qui progresse par l'alternance entre résolution de problèmes locaux et reconfiguration du problème global[39].

4.2 La programmation, conception exemplaire

Dans cette section, nous étudions le rôle essentiel que doit jouer la programmation dans l'avènement d'une science du *design*. Nous commençons par établir le caractère paradoxalement non prescriptif que doit avoir, selon Simon une logique de la *conception*, ce qui lui permet de s'appuyer sur les mêmes outils logiques que les sciences et les mathématiques en général : les *conceptions* sont des objets positifs disponibles en droit à l'analyse (§ 66). C'est l'informatique qui a néanmoins permis à cette possibilité de se concrétiser, grâce aux programmes qui sont l'expression achevée des méthodes (§ 67). Dans le dernier développement § 68, nous montrons à quoi cette science du *design*, fondée sur une informatique conceptuelle, devait concrètement ressembler dans l'esprit de Simon.

38. (Simon 1972, p. 72).
39. (Simon [1967] 1977, p. 312-317).

§ 66. Une logique descriptive pour des objets intentionnels

Le développement précédent permet de comprendre que la *conception* pourra se constituer en science si elle est capable de se prendre pour objet réflexivement, c'est-à-dire d'observer les méthodes et les espaces de problème comme des objets. Or une telle possibilité n'est pas elle-même évidente.

Simon lui-même avait conscience de cette difficulté, car ce savoir est « subtil » au sens propre, c'est-à-dire prompt à se subtiliser à l'observation :

> Le noyau du problème réside dans l'expression *science artificielle*. […] Une science des phénomènes artificiels court toujours un danger imminent de se dissoudre et de disparaître [*of dissolving and vanishing*]. Les propriétés particulières de l'artefact se trouvent dans la mince interface entre les lois naturelles en son intérieur et les lois naturelles à son extérieur[40].

De ce fait, l'invasion des méthodes des sciences classiques dans l'enseignement des professions suit une certaine pente naturelle :

> Les écoles d'ingénieurs sont progressivement devenues des écoles de physique et de mathématiques ; les écoles de médecine sont devenues des écoles de sciences biologiques ; les écoles de commerce sont devenues des écoles de mathématiques discrètes. L'utilisation d'adjectifs comme « appliquées » dissimulait le fait, mais ne le changeait pas. […] Cela ne signifiait pas que le *design* continuait d'être enseigné, *de manière distincte de l'analyse*[41]. [*nous soulignons*].

Le terme d'*analyse* fait ici référence à une interprétation originale de l'opposition classique entre analyse et synthèse, que Simon associe dès le début des *Sciences de l'Artificiel* à celle entre le caractère prescriptif des techniques et celui descriptif des sciences naturelles :

> *Synthétique* est souvent utilisé dans le sens plus large de *conçu* ou *composé*. Nous disons de l'ingénierie qu'elle a trait à la *synthèse*, tandis que la science a trait à l'*analyse* […] L'ingénieur, et plus généralement le concepteur, s'occupe de comment les choses *devraient* [*ought*] être – comment elles devraient être pour *atteindre des objectifs* et pour *fonctionner*[42].

Simon ramène ici l'opposition épistémique classique entre analyse et synthèse à celle entre être et devoir-être (*is /ought* en anglais), qu'on doit à Hume[43]. Cette distinction établit l'indépendance *de droit* des sciences de l'artificiel, car le *devoir-*

40. *SA*, p. 112.
41. *SA*, p. 111.
42. *SA*, p. 4-5.
43. Hume, *A Treatise of Human Nature*, livre III, première partie, section 1.

être désigne à l'inspection un point de vue sur les objets du monde qu'aucune science n'atteint, celui de leurs propriétés fonctionnelles. Cependant cette charge normative ne pose-t-elle pas problème ? se demande Simon. Peut-on qualifier de scientifiques des propositions qui devraient être formulées dans une logique modale ?

> La science naturelle a trouvé un moyen d'exclure le normatif et de s'occuper seulement de comment sont les choses. Pouvons-nous et devons-nous maintenir cette exclusion quand nous passons des phénomènes naturels aux phénomènes artificiels, de l'analyse à la synthèse[44] ?

À cet endroit, une note de bas de page renvoie à une discussion subséquente du chapitre 5 – qui renvoie lui-même, sur cette question, aux deux articles déjà cités – ainsi qu'au chapitre 3 d'*Administrative Behavior*. Ces textes soutiennent deux points qui peuvent sembler à première vue antithétiques :

1) Établir des faits sur les artefacts n'exige pas autre chose que la logique classique, non modale, du premier ordre[45]. Simon donne l'exemple d'un problème d'optimisation multi-contraintes :

> La logique est exactement la même que si nous devions adjoindre les contraintes liées à l'objectif et la demande de maximisation comme de nouvelles « lois naturelles », aux lois naturelles existantes incorporées à l'environnement. Nous demandons simplement quelles valeurs les variables de commande *auraient* dans un monde répondant à toutes ces conditions, et concluons que telles sont les valeurs qu'elles *doivent* avoir[46].

2) Le *devoir-être* n'est cependant pas *réductible* à une donnée de fait. Ici, Simon suit Ayer[47] et les empiristes logiques dans la thèse qu'on ne peut définir le bien, les valeurs, le devoir à partir de grandeurs naturelles comme le plaisir et l'utilité. Il ne va pas jusqu'à suivre Ayer lorsqu'il affirme que les propositions éthiques seraient de simples expressions émotionnelles ; il se contente de maintenir le domaine éthique dans une indépendance logique radicale des questions de fait :

> Des propositions de fait ne peuvent être dérivées de propositions éthiques par aucun processus de raisonnement, non plus que ces dernières peuvent être comparées directement avec des faits – puisqu'elles affirment des *devoir-être* plutôt que des faits[48].

44. *SA*, p. 5.
45. *SA*, p. 114-115.
46. *SA*, p. 118.
47. (Ayer [1936] 1952, p. 105).
48. (Simon [1947] 1997, p. 56).

Et, dans son commentaire de l'édition de 1997, où nous laissons les expressions anglaises *is / ought* dans le texte :

> Le point fondamental est que vous ne pouvez obtenir un « *ought* », quelle que soit la prudence de votre raisonnement, à partir d'un seul ensemble de « *is* ». Pour parvenir à un « *ought* » dans les conclusions, au moins un « *ought* » devait se cacher dans les prémisses initiales. Aucun cumul de connaissance concernant la manière dont le monde est réellement, en lui-même, ne nous dit comment le monde devrait être. Pour cela, nous devons accepter de dire quel type de monde nous voudrions avoir ; nous devons poser des valeurs qui vont au-delà des faits[49].

Étudier de manière scientifique les phénomènes artificiels revient donc à les purifier de leur composante déontique, à repousser celle-ci jusqu'à ses finalités les plus lointaines afin de mettre en avant les relations et les faits postifs qu'ils supposent. Simon reprend un passage du manuel de l'infanterie de l'armée américaine : « La surprise est un élément essentiel d'une attaque réussie[50] ». Dans cette proposition se cache une proposition « déontique » : « Réussis ton attaque ! » et une proposition technique : « Une attaque n'est réussie que lorsqu'elle est menée sous condition de surprise ». À partir de cette seconde proposition, on est dans le pur domaine positif de la *conception*, et on peut développer les moyens nécessaires à la surprise, comme l'incertitude concernant le temps et l'heure, l'évitement de procédures stéréotypées, la vitesse d'exécution, etc.[51]

Le *devoir-être* (*ought*) motive donc l'existence des problèmes, des procédures et des artefacts, mais ne permet pas de les constituer comme objets d'étude positifs. Cette possibilité (le point 1 ci-dessus) n'est apparue de manière systématique que depuis l'avènement de l'informatique. C'est elle donc qui permet au *design* de se constituer en science.

§ 67. Calcul et délibération

La science du *design* est rendue possible par l'avènement de l'informatique, car celle-ci la dote de la réflexivité qui lui est essentielle. Pour la première fois la *conception* est capable de *réfléchir* à ses objets : les productions de l'activité informatique – les programmes – sont à la fois des méthodes pures et des artefacts.

49. (Simon [1947] 1997, p. 68-69).
50. (ibid., p. 57).
51. *SA*, p. 58.

La citation déjà donnée au début de la section précédente[52] se prolonge ainsi :

> Il existe aujourd'hui un nombre considérable d'exemples de processus de *conception*, d'espèces très différentes, qui ont été complètement définis et fondus dans le métal, pour ainsi dire, sous la forme de programmes informatiques qui tournent réellement : optimisant des algorithmes, des procédures de recherche, et des programmes à visée spécifique pour concevoir des moteurs, équilibrer des lignes d'assemblage, sélectionner des portefeuilles d'investissement, situer des entrepôts, concevoir des routes, diagnostiquer et traiter des maladies, et ainsi de suite. Comme ces programmes décrivent des processus de *conception* complexes, dans un détail minutieux et complet, ils sont disponibles pour inspection complète et analyse, ou pour essai par simulation. Ils constituent un corpus de phénomènes empiriques auquel l'étudiant en *design* peut s'adresser et qu'il peut chercher à comprendre. Il n'est plus question, depuis que ces programmes existent, que le processus de *conception* se cache derrière le voile du « jugement » ou de l'« expérience »[53].

La science du *design* est donc une science *empirique*. Simon insiste d'autant plus sur ce point que cela peut sembler contre-intuitif, étant donné que l'objet de cette science sont des conceptions pures de l'esprit humain, et qu'on pourrait la penser être une discipline mathématique[54].

Simon place ainsi l'informatique au rang des progrès majeurs de l'histoire humaine. Il ne s'intéresse pas tant à la prouesse technique que représente l'ordinateur, qu'au progrès conceptuel qu'il annonce. Aussi ne le compare-t-il pas, par exemple, à l'électricité ou à l'aviation, ces autres découvertes techniques majeures du xx^{e} siècle, mais plutôt à l'écriture ou à l'algèbre :

> Dans les manuels d'histoire de la civilisation humaine, l'invention de l'écriture et l'invention de l'imprimerie sont toujours traitées comme des événements capitaux. Peut-être dans les futurs manuels d'histoire, l'invention des communications électroniques et l'invention de l'ordinateur seront mises en avant de manière comparable. Ce que tous ces développements ont en commun, et ce qui les rend si importants, c'est qu'ils représentent des changements élémentaires dans l'outillage qu'a l'homme pour faire des choix rationnels – dans ses compétences calculatoires[55].

On délibère en effet d'autant mieux qu'on réduit la décision à un calcul. Il ne s'agit pas là d'une pétition de principe. Simon ne part pas de la notion de calcul pour

52. Voir plus haut, § 63.
53. *SA*, p. 134.
54. (Simon [1967] 1977, p. 214).
55. (Simon 1978a, p. 14).

réduire la délibération au *design*, mais c'est plutôt son analyse de la délibération comme *design* qui le conduit à la notion de calcul. La référence à l'écriture et à l'imprimerie ici n'est pas fortuite, puisque le calcul doit ici être entendu dans le sens moderne que lui donne le programme formaliste de Hilbert, comme réécriture de formules selon des règles. La notion de *système de symboles* établit le lien entre calcul et délibération.

> [Les symboles] sont des motifs physiques (par exemple, des marques de craie sur un tableau) qui peuvent apparaître en tant que composants de structures de symboles (parfois appelés «expressions») […] Un système de symboles possède également un certain nombre de processus simples qui agissent sur ces structures de symboles, des processus qui créent, modifient, copient et détruisent les symboles. Un système physique de symboles est une machine qui, tout en se déplaçant dans le temps, produit une collection évolutive de structures de symboles. […] L'hypothèse est qu'un système de symbole physique du type de celui que je viens de décrire dispose des moyens nécessaires et suffisants pour mener une action intelligente en général[56].

Les « processus qui créent, modifient, copient et détruisent les symboles » peuvent évoquer la définition d'une machine de Turing. Nous pensons qu'elle est plus proche des systèmes de production* de Post, qui se décrivent exclusivement comme des systèmes de réécriture, et que Simon, sous l'influence de Newell, commence à voir dès la fin des années 1960 comme la forme la plus générale et la plus pure du calcul – et de la vie mentale en général.

Simon et Newell (1972, p. 44-46) décrivent informellement une *production* comme un couple « *Condition* → *Action* », la condition portant sur l'état du système en jeu, l'action venant modifier cet état. Elle n'est donc rien d'autre qu'un couplage entre un état et une action, c'est-à-dire l'objet élémentaire de la délibération[57]. Un système de production (ou système de symboles, dans la terminologie des *Sciences de l'Artificiel*), quant à lui, est un ensemble ordonné de telles règles, qui permet de déterminer des séquences de transformations d'états. Le formalisme des systèmes de production a la même puissance calculatoire que celui des machines de Turing. Que le formalisme de Post permette de définir la notion de calcul à partir de ce couplage a dû certainement frapper Simon et l'encourager à l'adopter. Une prise de décision rationnelle, ce n'est rien d'autre, *a minima*, que l'application d'une règle d'action spécifiée par un environnement. Agir rationnellement, dans ce cadre, n'est rien d'autre qu'exécuter un tel système de règles, et

56. *SA*, p. 22-23.
57. Voir plus haut, § 64.

délibérer, ce n'est rien d'autre qu'élaborer ce système, c'est-à-dire le *programmer*.

§ 68. Programmation et science du *design*

La programmation permet donc de fixer les découvertes de la *conception* comme des objets réutilisables par la suite, par d'autres entreprises de *conception*. Une analogie que prend Simon concerne l'apprentissage en mathématiques, qui est une sorte d'*auto-programmation* :

> Lorsqu'un être humain a appris à faire de l'algèbre, il ou elle peut résoudre une équation qui lui est présentée, comme on dit, pratiquement sans y penser. Lorsque l'être humain a un algorithme qui dit « Oh, maintenant je soustrais *4x* à gauche et à droite de l'équation, et maintenant je fais ci et maintenant je fais ça », à ce moment on peut dire que l'intelligence a été expurgée du processus [*that the intelligence has been squeezed out of the process*]. Toute l'intelligence a été mise dans l'invention puis dans l'apprentissage de l'algorithme. Cet algorithme devient une affaire de routine [*a matter of rote*] pour quiconque est expert dans l'art. C'est ce genre de chose qu'est un compilateur[58].

La référence aux compilateurs* dans le passage cité n'est pas accidentelle. Le texte dont il est extrait, une conférence donnée en 1986 à une audience d'informaticiens, est un des développements les plus complets qu'ait donné Simon sur sa vision de la programmation et son rapport à la connaissance. Un compilateur, dit-il, est un algorithme qui traduit des instructions écrites dans un langage de haut niveau en instructions élémentaires que la machine peut exécuter directement. Cette traduction est déterministe, c'est-à-dire qu'on sait exactement comment la machine doit se comporter à l'invocation de l'instruction de haut niveau ; il s'agit d'un problème qu'on sait résoudre, comme l'étudiant sait résoudre l'équation d'algèbre. Il n'y a plus d'intelligence dans cette activité, et c'est pour cela qu'elle peut être automatisée. Cela permet au programmeur de se concentrer sur un problème difficile, pour lequel on ne possède pas encore d'algorithme ou pour lequel il n'en existe pas de satisfaisant. Programmer, c'est trouver de telles solutions satisfaisantes ; le programme qui en résulte peut ainsi donc être réutilisé face à des problèmes similaires ; lorsque les cas de figure varient, il est sans doute possible de factoriser des portions du programme qui résolvent partiellement le problème dans ses dimensions générales. Ces solutions générales se présentent comme des fonctions qui peuvent être intégrées au compilateur sous forme de librairies* prêtes à l'emploi.

58. (Simon 1986, p. 727). Voir également (Simon 1978b, p. 17).

On peut donc s'attendre à ce que la part d'automatisation augmente avec le temps, et que la tâche du programmeur reste focalisée sur les problèmes nouveaux, ou les situations nouvelles, ou les combinaisons nouvelles de problèmes et de situations.

> Nous avons déplacé cette frontière de l'automatisation. Nous avons placé une charge croissante des tâches sur les ordinateurs[59].

Cependant cela n'est qu'une première étape. Le défi pour Simon est de déplacer cette frontière encore plus loin, en visant la formalisation même du problème et la recherche de solution :

> Ce que les compilateurs ne font pas, c'est résoudre le problème de convertir la situation qui est présentée sous une forme non structurée, comme une description informelle du problème. Le compilateur ne réalise pas la tâche de convertir cette description en quelque chose à partir de quoi l'un de ces algorithmes systématiques, peut-être complexes, peuvent travailler[60].

Simon ne s'intéresse donc pas à augmenter l'efficacité du codage, mais bien à insérer l'ordinateur dans le problème réel auquel s'attaque le programmeur, qui consiste à le décrire ainsi que sa solution en éléments actionnables, ce qui est la tâche de la spécification. Le point clé, ici, est qu'avant même de s'atteler à la tâche de trouver une solution satisfaisante dans l'espace des possibles, il faut poser cet espace, passer d'une situation non-structurée à une « situation formalisée[61] » – chose que seuls les humains savent faire. Pour Simon, cela ne se fait pas différemment que pour un problème bien posé ; on applique des heuristiques à des méta-espaces de problèmes : « dans telle situation (informelle), essaie telle représentation (formelle) du problème ». Il ne pense donc pas que chaque situation informelle que l'on rencontre requiert une opération « magique » de l'entendement pour la formaliser, mais qu'une large part de ce travail peut être décrite par des procédures – par exemple des entretiens avec le « client » pour comprendre et clarifier son besoin, l'identification des situations ambiguës, des exceptions, etc. Ces procédures, comme les heuristiques de recherche de solution, n'offrent jamais de garantie de solution, mais elles forment des méthodes qui peuvent être au moins partiellement automatisées.

Simon résume tout ce raisonnement en identifiant trois bases de connaissances qui doivent permettre aux ordinateurs d'assister les personnes dans leurs entre-

59. (Simon 1986, p. 730).
60. (ibid., p. 727).
61. (ibid., p. 727).

prises de *design* :

1. Une base de procédures déjà automatisées et mobilisables pour la résolution de problème. Ce « patrimoine procédural » (dans le langage d'aujourd'hui : librairies de fonctions ou de services), comme on pourrait l'appeler, permet de représenter les opérateurs disponibles de l'espace de problème.
2. Une base de représentations de données (*data representations*) qui permet de représenter les problèmes eux-mêmes, c'est-à-dire les états possibles de la situation, ses contraintes, les objectifs recherchés.
3. Une base de « stratégies » ou « procédures de plus haut niveau[62] » – les procédures (C) discutées plus haut[63].

Simon ajoute à ces trois réquisits le besoin d'un langage de spécification formel, qui serait une sorte de *lingua franca* fournissant un cadre d'expression à ces bases, et dans lequel on pourrait donc réaliser ce travail de formulation de problèmes et de solutions[64].

Un langage de programmation, qui propose *a minima* un ensemble cohérent des deux premières bases de connaissance, est donc une théorie des représentations ou encore des espaces de problème, puisque les deux termes sont équivalents[65].

C'est donc dans les langages de programmation que devrait le mieux se voir le progrès de la théorie des représentations qui, selon Simon, manque aujourd'hui le plus à la science du *design*. Le passage à ce sujet, cité plus haut, continue ainsi :

> Mais même si notre classification est incomplète, nous commençons à construire une théorie des propriétés de ces représentations. Les théories en expansion des architectures d'ordinateurs et des langages de programmation –- par exemple, les travaux sur les langages fonctionnels et sur les langages orientés-objet –- illustrent certaines directions qu'une théorie des représentations peut prendre[66].

Cette phrase fait référence au constat, commun dans les années 1990, que les différentes familles de langages de programmation – les langages fonctionnels, les langages objet, la programmation logique, etc. – modifient radicalement la relation du programmeur à son problème, c'est-à-dire la puissance d'expression qu'il peut en avoir, ses heuristiques de résolution, voire les algorithmes qu'il privilé-

62. (ibid., p. 729).
63. Voir plus haut, § 64.
64. (Simon 1986, p. 730).
65. (Kaplan et Simon 1990, p. 376).
66. *SA*, p. 134.

giera[67]. On peut noter que l'idée d'une théorie des représentations se trouve bien déjà dans l'édition originale de 1969, même si Simon paraît moins assuré à son propos.

Ce que Simon décrit ainsi dans cette vision des trois bases de connaissances du *design*, c'est une dynamique cumulative de la connaissance pratique rendue possible par les ordinateurs : ceux-ci devraient permettre en effet de formaliser les connaissances procédurales, représentationnelles et heuristiques en bases de données qui permettraient en retour aux personnes d'automatiser ou semi-automatiser le traitement des problèmes. Elles pourraient alors se concentrer sur des problèmes plus difficiles – et capturer ainsi de nouvelles procédures, mettre au jour de nouvelles représentations et découvrir de nouvelles heuristiques qui viendraient à leur tour enrichir ces bases.

Il ne s'agit donc pas de « mécaniser » l'intelligence et la *conception*– au moins pas à l'horizon visible, mais de construire une dynamique interactive entre l'homme et la machine sur laquelle Simon insiste particulièrement. Il prend ainsi l'exemple du chimiste E. J. Corey, qui obtint le prix Nobel notamment grâce à l'introduction de méthodes computationnelles pour calculer des chemins de synthèse possibles de molécules organiques.

> Si vous jetez un œil au travail que Corey a fait ces dernières années, vous verrez que ce n'est pas basé de manière primaire sur de la *conception* automatique, car dans ce cas on aurait donné le prix Nobel à l'ordinateur, mais sur de la *conception* interactive où l'ordinateur et le chimiste jouent tous deux un rôle substantiel[68].
>
> Dans cet article je veux parler du logiciel (*software*) comme de l'esprit du système [par opposition au *hardware*], mais pas son esprit exclusif, car les personnes qui développent le système et ceux qui utilisent le système, bien sûr, contribuent aussi largement à cet esprit[69].
>
> [à propos du système téléphonique:] Le système a toujours été un système homme-machine. Des moyens ont été trouvés de déplacer cette frontière imperceptiblement d'un système qui était presqu'entièrement composé de femmes et d'hommes à un système aujourd'hui qui est presqu'entièrement composé d'ordinateurs[70].

67. Voir Backus (1978) et R. W. Floyd (1979) et (Priestley 2017, p. 453-455) pour l'histoire des notions de *style* et de *paradigme* de programmation.
68. (Simon 1994, p. 6).
69. (Simon 1986, p. 727).
70. (ibid., p. 731).

La science du *design* doit donc permettre une nouvelle symbiose entre l'homme et la machine[71], un déplacement d'une interface au sein d'un système global d'intelligence qui d'ores et déjà dépasse les limites de l'esprit humain.

Que ces effets n'apparaissent pas aux acteurs de ce système, et qu'ils soient dubitatifs quant à l'intelligence artificielle agaçait Simon. L'automatisation de la *conception* progresse, mais elle devient aussitôt invisible. Ainsi dit-il dans une conférence donnée à des étudiants en *design* :

> Si nous faisions un tour de la salle je suis sûr que nous aurions un récital d'exemples de programmes qui participent au processus de la *conception* […] J'ai trouvé à Cornell il y a quelques mois un programme qui était tout à fait qualifié pour concevoir des processus élémentaires d'usines chimiques. Dans la salle vous pourriez donner beaucoup d'exemples de *conception* automatique ou semi-automatique de circuits ou de processeurs et de je ne sais quoi encore[72].

Si la programmation est donc la forme générale de la pratique de la *conception*, et qu'elle permet de mettre au jour et de répertorier, de manière directement opérables, les diverses formes de représentations, les procédures élémentaires, et les stratégies de chaque domaine de *conception*, on comprend qu'elle permette au *design* de se constituer en *science*. C'est le sens de cette proposition, qui est le projet tout entier des *Sciences de l'Artificiel*, que nous voulons à présent explorer. Car, comme nous l'avons relevé en introduction, n'est-il pas paradoxal de vouloir théoriser la connaissance pratique ?

4.3 Critique des thèses d'H. Simon

Ayant exposé l'interprétation par Herbert Simon de la nature de la connaissance pratique et du rôle exemplaire que doit y jouer la programmation, nous souhaitons dans cette section en proposer une évaluation critique. Nous commençons par évaluer la postérité du projet des sciences de l'artificiel. La science du *design* s'est bien développée suite à l'appel de Simon, cependant c'est dans des directions tout autres que celles qu'il appelait de ses vœux (§ 69), et si l'usage de l'intelligence artificielle progresse dans ces disciplines c'est grâce à l'apprentissage machine*, une méthode à laquelle Simon était hostile. Il ne s'agit pas là simplement d'un échec de politique scientifique. L'idée programmatique d'une science

71. (Simon 1976a).
72. (Simon 1994).

générale du *design*, telle que la conçoit Simon, conduit à celle d'une Science Générale au sens leibnizien, à laquelle toutes les sciences naturelles et les arts seraient subordonnées (§ 70).

On peut s'étonner d'un tel résultat. L'idée que la résolution de problème est *conception* d'interfaces entre un agent régulé et une situation est attrayante et semble correspondre à l'interprétation contractuelle que nous avions donnée de la programmation au chapitre 3. Cette idée centrale est en quelque sorte défigurée, chez Simon, par son soutien inconditionnel à l'hypothèse computationnelle de l'esprit, qui le conduit à souscrire également à l'interprétation classique de la programmation comme réglementation d'opérations symboliques (§ 72). Cependant, même délestée de cette hypothèse, l'idée que la programmation serait la forme générale et le modèle de la résolution de problème rencontre d'autres difficultés que nous illustrons par l'étude de l'algorithme de Deep Blue, le premier ordinateur qui vainquit un champion du monde d'échecs (§ 71). Ce dernier développement introduit la question directrice du chapitre suivant, concernant la spécificité de la programmation au sein de la résolution de problème.

§ 69. La postérité des sciences de l'artificiel

Soixante ans plus tard, peut-on dire que le programme des *Sciences de l'Artificiel* est avéré ? Assiste-t-on à l'émergence d'une science du *design* ? Les professions se trouvent-elles reconfigurées par la programmation ? Les bases de connaissances que Simon voyait se constituer jouent-elles le rôle universel qu'il leur assignait dans la résolution de problème ?

Sur toutes ces questions, les réponses sont mitigées. L'étude du *design*, dans sa composante méthodologique, est devenue une discipline universitaire à part entière, mais la référence à Simon, si elle y est importante, est empreinte d'un regard critique. Des approches plus dialectiques, interrogatives, qui donnent plus d'importance à l'expérience vécue devant l'objet et à son histoire, se sont développées à partir du mouvement des *Design Methods*, contemporain de la réflexion de Simon[73].

Au crédit de la position de Simon, il y a la transformation majeure que l'informatique suscite dans toutes les professions, sur laquelle nous avons déjà maintes fois insisté. L'informatique amène les professions à donner de nouvelles solutions

73. Sur ces sujets, voir notamment (Buchanan 2009) et (Cross 2006).

de problèmes anciens, à les reformuler radicalement, ou à se poser des problèmes nouveaux. Les langages de programmation, comme l'observait Simon, sont devenus de plus en plus spécialisés et aptes à décrire des classes précises de problèmes selon leur logique propre, ainsi que nous l'avons vu avec Modelica et Catala[74]. Les logiciels de CAO*, quant à eux, intègrent toujours plus de tâches de conception initialement réservées aux personnes humaines, ainsi qu'il le prédisait[75].

Il existe cependant de nombreux écarts entre cette situation que nous pouvons observer et la vision programmatique de Simon. D'abord, il ne semble pas qu'on puisse observer aujourd'hui de convergence des méthodes et des représentations computationnelles propres à chaque discipline en direction d'une science universelle du *design*. Il y a certes le mouvement pédagogique de la pensée computationnelle (*computational thinking*), mentionné en introduction[76], qui tente d'inculquer aux étudiants une culture générale concernant les concepts et les méthodes clés de la programmation. Cependant le contenu positif de ce savoir et son organisation sont encore assez flous[77]. Des taxonomies, pour reprendre le mot de Simon, commencent à émerger, mais elles restent empiriques et assez génériques – on n'est pas encore capable de connecter les procédures et les représentations qu'elles décrivent à leurs applications spécialisées au sein de chaque discipline. Surtout, il manque de base théorique pour en donner raison. Lorsque Denning, Martell et Cerf (2015) tentent de répertorier les « grands principes de l'informatique » ils aboutissent à une liste empirique et ouverte d'une quarantaine de propositions, dont on ne voit pas très bien l'unité systématique. Les langages de programmation, quant à eux, peuvent être vus comme des théories *de facto*, mais leur classement lui-même reste problématique, tout autant que celui des paradigmes de programmation*, comme la programmation fonctionnelle* ou la programmation logique*, plus abstraits.

Par ailleurs, ces transformations ne concernent pas tous les champs de la rationalité pratique, dont certaines restent pour l'instant inaccessibles à la programmation. Dans la gestion des affaires économiques, les ordinateurs restent cantonnés à des tâches ou à des décisions spécialisées, et ne parviennent pas à participer de manière significative à la prise de décision de haut niveau.

74. Voir plus haut, § 37.
75. Voir là-dessus, dans le domaine de l'architecture, (Gaudillière-Jami 2022). Nous développons quelque peu ce sujet en § 217.
76. Voir plus haut, § 10.
77. (Bocconi et al. 2016, p. 6).

De manière plus technique, la vision qu'avait Simon de « compilateurs intelligents », qui intégreraient la fonction d'analyse et de formulation des problèmes, a certes fait des progrès spectaculaires de manière très récente[78]. Cependant ces progrès ne s'appuient pas sur la rationalité procédurale promue par Simon, mais bien plutôt sur la simulation statistique de fonctions de décisions obtenue par apprentissage machine*.

Or Simon était très sceptique quant à l'importance de l'apprentissage machine* statistique, et en particulier à celle des réseaux de neurones[79]. La seule forme d'apprentissage machine* qu'il promeut, et qu'il associe plutôt à la découverte de nouveaux savoirs, est l'induction de règles[80], qui contraint un algorithme statistique à présenter les corrélations trouvées entre séries de données sous forme de relations propositionnelles, comme *« Si le député a voté pour le budget, contre le gel des salaires et pour la loi sur l'éducation, alors il est (doit être) démocrate »*[81] – une contrainte dont se passent la plupart des algorithmes d'apprentissage machine* modernes. En d'autres termes, pour Simon la rationalité pratique doit nécessairement se formuler sous forme de règles unitaires, c'est-à-dire de productions au sens de Post, comme nous l'avons vu plus haut. Il s'agit là d'une condition élémentaire de la *discursivité*. Comme les prédictions des réseaux de neurones ne peuvent être ainsi formulées, ceux-ci sont appelés *sous-symboliques* (*sub-symbolic*) dans les débats des années 1980 à leur sujet. L'orientation actuelle des sciences et des professions vers l'usage intensif de l'apprentissage machine* statistique tourne donc le dos à la vision « symbolique » – il vaudrait mieux dire *discursive* – de Simon.

§ 70. L'hégémonie douteuse de la science de la *conception*

On peut objecter à l'argument du développement précédent que, si la vision programmatique de Simon n'est pas encore réalisée, cela ne l'invalide pas en droit. Certaines grandes idées ont un temps long de maturation, et on peut s'appuyer sur l'importance croissante que prend l'informatique dans tous les domaines de la vie pratique et toutes les disciplines pour penser que « la théorie finira bien par suivre », c'est-à-dire que la structuration et le contenu des disciplines universitaires

78. Voir plus haut, § 25.
79. Voir par exemple son jugement sur le premier modèle de réseaux de neurones, le Perceptron, (Simon 1983, p. 32).
80. (Langley et Simon 1995).
81. (Džeroski et al. 1997, p. 98).

liées aux professions et au *design* convergeront, dans le temps long, vers les idées de Simon. À ce titre, l'intégration croissante de l'architecture à l'ingénierie *par* l'informatique pourrait donner une idée de cette convergence[82].

La question est donc celle-ci : peut-on souscrire *en droit* à l'idée d'une *science générale du* design et de *sciences de l'artificiel* qui en seraient les déclinaisons pour chaque profession ? Il s'agit là en effet d'une proposition qui semble répondre à notre question directrice concernant la programmation. Cette activité serait en effet exemplaire, en ce qu'elle serait non seulement une profession parmi les autres, mais également la forme dans laquelle toutes devraient ultimement exprimer leurs procédures et leurs problèmes de manière à en faire un objet de science et à permettre la circulation du savoir entre elles.

Cependant, chez Simon cette idée est brouillée, notamment quant à sa distinction avec la science « classique », pour trois raisons au moins : D'abord, le domaine des sciences de l'artificiel semble s'étendre bien au-delà du périmètre des professions, et aller jusqu'à annexer de larges pans des sciences naturelles. Ensuite, toute investigation scientifique elle-même apparaît comme une activité de conception, ce qui fait de chaque théorie un *artefact*. Enfin, celles-ci se retrouvent par là-même paradoxalement dotées de propriétés performatives. Nous allons développer ces trois points successivement.

Reprenons d'abord l'idée que le *devoir-être*, (*ought*) donne son objet à la science du *design*. Par exemple, il s'agit de déterminer comment nous pourrions voler. On peut réexprimer ce problème en recherchant quel objet volant pourrait nous transporter, et c'est cela que nous appelons *avion*, de manière problématique. Il s'agit d'objets matériels qui satisferaient certaines conditions phénoménales dans un environnement qui est ici l'atmosphère terrestre. C'est pour cela que l'objet technique est essentiellement une *interface*, car il établit une distinction au sein même de l'environnement matériel, entre un environnement interne auquel nous identifions l'artefact et un environnement externe auquel il doit s'adapter.

Cependant, le naturaliste adopte exactement la même perspective que l'ingénieur lorsqu'il se demande comment volent les oiseaux : il étudie ce phénomène comme s'il s'agissait d'un artefact qui doit avoir cette fonction[83]. Les sciences de l'artificiel de Simon engloberaient donc une bonne partie des sciences de la vie,

82. (Gaudillière-Jami 2022).
83. *SA*, p. 6-7.

dont la recherche est guidée par des questions fonctionnelles, où plus exactement par des questions relatives à l'adaptation des organismes à leur milieu[84]. Pour les mêmes raisons, Simon affirme que la psychologie, l'économie, la sociologie, la linguistique sont également des sciences de l'artificiel[85]. On voit ici le brouillage de l'idée originale d'une approche scientifique propres aux professions, car il s'agit bien là de sciences qui, si elles ne sont certes pas des « sciences naturelles » au sens strict, prennent la même posture théorique qu'elles quant à leurs objets.

Rien n'empêche dans cette perspective d'étendre encore plus le domaine des sciences de l'artificiel. Car de nombreuses questions de sciences naturelles pourraient être dites adopter cette même posture – par exemple celles qui expliquent l'émergence d'un phénomène dynamique stable. Se demander pourquoi un tourbillon *doit apparaître* dans telle situation n'est pas très différent de se demander comment en *faire apparaître* un. Le dernier chapitre des *Sciences de l'Artificiel*, qui affirme que les mêmes « architectures de la complexité » sont à l'œuvre dans les systèmes sociaux, biologiques et physiques, suggère fortement une telle conclusion. Les méthodes et les concepts des sciences de l'artificiel se révèlent d'une fécondité qui s'étend bien au-delà des artefacts humains. De l'idée que le domaine des sciences de l'artificiel était si subtil qu'il était resté largement inaperçu, le voici qui à présent prend une ampleur considérable.

Cela est encore accentué par un second rôle que Simon confère à ce domaine, celui d'assurer une liaison entre les cultures scientifique et humaniste, entre la rigueur de la connaissance et la créativité de l'art. En effet, une science du *design* exposerait « le cœur commun de toute connaissance[86] » qui est la créativité dans la résolution de problème[87]. Faire de la science vivante, c'est essentiellement découvrir et inventer. Il s'agit bel et bien, selon Simon, d'activités de *conception*, dont les artefacts sont ici les théories elles-mêmes.

Cela implique en particulier que la science du *design* est également une logique de la découverte scientifique, que Simon affirme possible contre Popper. Simon consacre à ce thème une partie importante de sa recherche au cours des années 1980 et 1990[88], y voyant le champ le plus difficile de l'heuristique, où il est fait le

84. Voir (Simon 1978a, p. 3-4) pour la critique de la notion de *fonction*, qu'il cherche à remplacer par celle d'adaptation.
85. *SA*, p. 30, 77, 186.
86. *SA*, p. 136.
87. Simon revient à de nombreuses reprises sur ce point – par exemple (Simon 1982).
88. Voir notamment (D. Kulkarni et Simon 1988 ; Langley, Simon et al. 1987 ; Simon 1984 ;

plus souvent appel aux notions de « génie » et d' « intuition » qu'il veut reconduire à des éléments objectifs, c'est-à-dire computationnels.

La démarche de Simon pour ce faire est double : d'une part étudier le procès historique de certaines découvertes scientifiques, comme celle de la loi de la radiation par Planck, et d'autre part développer des programmes qui reproduisent de telles découvertes, comme BACON qui, à partir d'un jeu de données expérimentales, propose des lois de comportement des phénomènes en introduisant des grandeurs théoriques explicatives.

La science du *design* étudie l'ensemble du « contexte de la découverte » et fait donc pendant à la logique stricte de la science qui expose les lois naturelles dans le « contexte de la justification ». Du même coup ce que Simon appelle *science naturelle* se révèle limité à un corpus théorique formalisé, à ce qu'on pourrait appeler *le texte idéal de la science*[89]. Cette vision est peut-être conforme à celle qu'en avaient les empiristes logiques, dont Simon s'est toujours réclamé[90]. Mais l'investigation scientifique proprement dite, ainsi que l'ensemble des pratiques scientifiques qui *produisent* ces artefacts que sont les textes idéaux de la science, sont dans ce cadre des activités de *design*, et relèvent donc des sciences de l'artificiel. Le statut modeste que Simon semblait réclamer pour les sciences de l'artificiel au début de son ouvrage, comme de « petites sœurs » des sciences naturelles, se révèle doublement formidable : à la fois donnant leurs objets et leurs méthodes à une bonne partie de ces sciences, mais également permettant décrivant la logique même de toute recherche scientifique.

D'où provient un tel renversement ? D'une certaine manière, Simon retrouve la veine méthodique des Temps modernes, qui cherchaient dans la Méthode le fil conducteur de la découverte scientifique[91]. Mais il ne s'agit plus ici, comme chez Descartes, de préceptes généraux devant guider la réflexion. Simon cherche bel et bien à établir une science de l'invention, un *ars inveniendi* ou encore d'une *heuristique*, cette science hypothétique que Joachim Jungius, au début du Grand Siècle, avait placé au sommet de l'édifice de la connaissance humaine[92], et que Leibniz évoque avec approbation.

Simon, Langley et Bradshaw 1981).

89. Nous empruntons cette expression à (Railton 1981), qui l'introduit dans le cadre de l'empirisme logique.
90. (Simon 1996a, p. 44).
91. Voir plus haut, § 13.
92. (Hügli et Theissmann 1971 ; Schepers 1971).

Cette seconde confusion en entraîne une troisième. Dans les sciences de l'artificiel, la théorie ne se contente pas seulement de fournir les principes de description et d'explication d'un champ de phénomènes, mais permet la *production* des phénomènes eux-mêmes :

> La théorie réalise [*performs*] la tâche qu'elle explique. C'est-à-dire, une bonne théorie de traitement de l'information concernant un bon joueur d'échecs humains peut bien jouer aux échecs ; une bonne théorie de comment les humains écrivent des romans écrira des romans[93].

Il n'y a plus ici de distance entre le phénomène et sa description. On ne dit pas, par exemple, que l'algorithme de la fin et des moyens présenté dans le programme GPS[94] est une *description approchée*, ou une *simulation* des processus de pensée des humains face à un problème pratique ; on dit plutôt que les processus de pensée humains *sont* l'exécution d'un tel algorithme (ou d'un algorithme qui ne diffère du premier que par des différences parasitaires, comme la limitation mémorielle, la fatigue, l'émotion, etc.). En particulier, programmer est ce qui permet de comprendre les phénomènes de la connaissance, comme la compréhension, l'invention, l'expertise, etc. Les recherches cognitives de Simon s'appuient sur des programmes informatiques – GPS, ISAAC, BACON, MATER, etc., et il comme le dit explicitement :

> Aujourd'hui, c'est un fait que nombre de théories en psychologie sont écrites comme des programmes d'ordinateur[95].

Cette confusion découle de l'hypothèse computationnelle de l'esprit, dont Simon fut l'un des plus ardents promoteurs. Les ordinateurs sont pour lui (comme les cerveaux humains) des « systèmes de symboles » régis par des règles purement syntaxiques, c'est-à-dire des *systèmes de traitement de l'information*[96]. Cette confusion peut être rendue manifeste par la double réponse que Simon ferait sans doute à la question *« Où est le programmeur ? »*

D'une part, dans l'idée d'une science du *design*, le programmeur est clairement en surplomb de la machine et de la situation dont il conçoit les interfaces et qu'il agence de manière à résoudre son problème. D'autre part cependant, dans l'hypothèse computationnelle de l'esprit, le programmeur est lui-même une machine qui traite les informations provenant de la situation selon des programmes prédéter-

93. (Simon et Newell 1972, p. 10-11).
94. Voir l'annexe A pour l'explication de ce programme célèbre.
95. (Simon 1978b, p. 14).
96. (Simon et Newell 1971, p. 148).

minés. Pour Simon, il n'y a là nulle contradiction, simplement une différence de point de vue et d'abstraction. Car, comme nous l'avons vu, la conception d'espaces de problèmes et la recherche d'algorithmes de résolution sont également pour lui des calculs procédant dans des méta-espaces de problème. Ainsi, une théorie psychologique est performative (« une bonne théorie de comment on écrit des romans écrira des romans ») *parce que* l'objet de la théorie, l'esprit humain, est lui-même un système de symboles exécutant des procédures.

On peut regimber devant cette vision trop intellectualiste des compétences humaines, qui suppose que le corps propre de l'agent est similaire à une machine se contentant d'obéir à des instructions prédéterminées, en partie par l'agent lui-même, et en partie par la sélection naturelle. Simon, on l'a vu, répondrait que, malgré les apparences, *c'est en effet le cas*. Un processus d'apprentissage est bien, de manière sous-jacente, un travail d'auto-programmation, c'est-à-dire de reconfiguration rapide par l'individu agent de ses réponses aux sollicitations de son environnement, de la même manière que la sélection naturelle reprogramme les espèces sur le long terme[97].

§ 71. L'illisibilité des livres de procédures

Notre seconde grande critique de la position de Simon concerne l'idée que la programmation serait la forme achevée de la résolution de problème. Une telle idée est, selon nous paradoxale. Pour cela, il faut se représenter ce que serait la description exhaustive du savoir positif d'une pratique sous la forme d'un livre de procédures – un peu comme l'immense répertoire permettant à John Searle, dans sa chambre chinoise, de tenir n'importe quel dialogue dans cette langue[98].

Il est utile de partir de l'exemple du droit, où une telle conception du savoir est possible, puisque ces livres de procédures existent – ainsi les différents codes de lois concernant le droit. L'organisation du droit en spécialités (droit commercial, droit social, droit immobilier, etc.) avec leurs codes spécialisés attenants, manifeste le défi concret de rendre accessible le contenu immense du savoir accumulé : l'organisation des lois suit de manière assez naturelle la typologie des problèmes rencontrés. Ainsi, s'il est question d'un conflit employeur - employé, c'est un spécialiste du droit du travail qui se saisira du dossier. Le droit organise également les priorités entre les différents codes si plusieurs sont applicables au litige : par

97. (Simon 1980, p. 36).
98. (Searle 1980).

exemple, dans le cas d'un conflit du travail impliquant des violences, le code pénal prend la priorité sur le code du travail.

Nous ne voulons pas affirmer ici que *tout* le savoir d'un juge ou d'un avocat réside dans ces codes et leurs jurisprudences. La nature épistémique de l'interprétation requise de ces textes pour produire une plaidoirie ou un jugement est une question très débattue dans l'épistémologie du droit, dans laquelle il n'est pas besoin d'entrer ici.

Nous voulons seulement remarquer qu'une telle conception est possible parce que le droit parvient à *organiser* son savoir positif en livres de procédure. Si une telle organisation n'était pas possible et que le droit se présentait sous la forme d'une liste immense et désordonnée de règles, la pratique du droit, telle que nous la connaissons aujourd'hui, et même son élaboration, serait impossible. Un livre de procédure ne suffit pas pour consigner et transmettre le savoir humain. Il faut aussi en donner le *mode d'emploi*, c'est-à-dire expliquer la manière d'accéder aux règles pertinentes en fonction du problème posé et gérer les hiérarchies de règles lorsque plusieurs peuvent s'appliquer. Or il est aisé de trouver des exemples de pratiques où, même si toutes les règles utiles pouvaient en être énoncées, un tel ordre et une telle hiérarchie seraient problématiques.

Les jeux de stratégie fournissent un bon exemple de ce problème. Chaque situation au jeu d'échecs ou de go est un problème à résoudre, mais il n'est pas possible de classer ces problèmes par typologie, à part celle, très élémentaire, de problèmes « de début, de milieu et de fin de partie ». Les joueurs classent les problèmes en reconnaissant des schémas (*patterns*) typiques sur le damier. Un expert aux échecs, selon Simon, dispose de plus de 50000 schémas de positions qu'il sait reconnaître immédiatement sur un échiquier, un peu comme un lecteur lit directement des mots entiers sur la page, sans épeler mentalement les lettres qui les constituent[99]. Chacun de ces schémas est bien une règle, même si elle est mémorisée de manière diagrammatique. Le joueur l'associe par exemple à un coup précis, ou encore à un objectif tactique, ou tout simplement à une évaluation positive ou négative de la situation.

Imaginons qu'un Grand Maître parvienne à consigner toutes ces règles dans un ouvrage, et même qu'il trouve un principe de classement astucieux permettant de retrouver aisément chaque règle pertinente pour une situation donnée (par exemple : *« règles où le Roi est sur sa ligne initiale »*, *« règles où la Dame est au*

99. (Simon 1984, p. 3).

centre de l'échiquier », etc.). Il restera à donner au lecteur des règles additionnelles permettant de hiérarchiser les règles applicables entre elles, selon des modalités peut-être complexes, comme « *tel schéma est mauvais, sauf si tel autre schéma est également le cas, alors un objectif de mat est possible* ».

Une telle configuration n'est pas imaginaire ; elle décrit à peu près, de manière très simplifiée, le fonctionnement du programme Deep Blue, qui vainquit Kasparov aux échecs en 1997[100], sans doute l'un des sommets de l'intelligence artificielle symbolique. En particulier, le programme était doté d'une fonction d'évaluation constituée de plus de 8000 paramètres décrivant des schémas allant du plus simple (telle pièce sur telle case) au plus complexe (tour sur ligne ouverte, protection du roi). L'importance accordée à chaque paramètre était elle-même influencée par différents facteurs, comme le nombre de pièces restant sur l'échiquier, ou un autre schéma. Le schéma de la tour sur ligne ouverte, par exemple, est décrit par 192 paramètres, et son importance est partiellement interdépendante avec celui de la protection du roi[101].

On peut arguer que Deep Blue est bien un « livre de procédures » pour jouer aux échecs, puisqu'il permet indéniablement de codifier, de manière absolument positive, les solutions des problèmes posés sur l'échiquier.

Cependant un tel argument pose un double problème. D'une part un tel livre serait proprement *monstrueux* à appliquer pour une personne humaine, et on arrive ainsi la notion paradoxale d'un *savoir non-humain* dont le sujet (l'ordinateur ?) est problématique. D'autre part, il reste encore *trop humain*, en ce qu'il encode des schémas répertoriés par l'expérience de joueurs traditionnels, avant de les recombiner par des pondérations estimatives. Il est significatif à ce titre que, entre chaque partie contre Kasparov, les coefficients de certaines sous-fonctions d'évaluation étaient ajustés manuellement par les programmeurs, selon leur analyse de la partie précédente. Éliminer ce bricolage revient à pousser à son terme la logique dans laquelle s'était engagée la fonction d'évaluation de Deep Blue c'est-à-dire, plutôt que fixer par avance la bibliothèque de schémas pertinents et leurs lois de priorité et de combinaison, laisser libre cours à l'association statistique des situations rencontrées et de leurs conséquences pour identifier les schémas à considérer ainsi que leur importance. Il se pourrait par exemple que, étant donné un certain schéma applicable à la situation, deux autres schémas distincts doivent être com-

100. (Campbell, Hoane et Hsu 2002).
101. (ibid., p. 73-76).

binés ensemble pour en tirer une règle pertinente ; ou encore, qu'il faille ajuster tel schéma en lui ôtant une pièce devenue indifférente, avant de le recombiner à un autre schéma auquel on aurait ajouté telle autre pièce, et ainsi de suite. C'est de cette manière qu'on peut décrire, de façon imagée, le fonctionnement d'un réseau de neurones, dont les phases d'apprentissage servent à associer entre eux différents schémas de position, selon des pondérations relatives, en tenant compte de leur seule pertinence statistique, et tout cela à plusieurs niveaux d'abstraction.

En d'autres termes, il semble logique (avec le recul de l'histoire) que le connexionnisme* ait été, pour ce type de problèmes, l'aboutissement de l'intelligence artificielle symbolique. Lorsque la logique humaine d'énonciation, de classement et de recombinaison des règles pertinentes est portée à sa limite, elle laisse place à leur auto-calibration statistique. C'est ainsi que le champion du monde de go a été vaincu, en 2016, par un programme construit entièrement selon cette logique, AlphaGo.

Un réseau de neurones est bien un livre de procédures, et il est tout autant symbolique que n'importe quel autre algorithme mathématique, comme se plaît à le rappeler Simon[102]. Pourtant il serait encore plus monstrueux à appliquer pour des personnes humaines que l'algorithme de Deep Blue. Les calculs qu'il commande sont non seulement d'un longueur qui nous est inaccessible, mais ils sont également complètement opaques à toute compréhension.

Il semblerait bien, au moins pour les pratiques affrontant des situations « enchevêtrées », comme c'est le cas des jeux de stratégie, que l'explicitation exhaustive des règles pertinentes à leur exercice aboutisse à un livre de procédures inaccessible à l'intelligence humaine. Ce n'est donc certainement pas le savoir pratique *humain* en question qui y serait représenté. On comprend mieux, par là, pourquoi Simon était instinctivement hostile à l'apprentissage machine* statistique : c'est que cette approche remettait en cause l'idée même d'une science générale du *design* et d'une rationalité procédurale c'est-à-dire, au fond, l'idée que le savoir *positif* qui a cours dans les professions et dans les pratiques peut en général être réduit à une somme de règles techniques.

Il y a là un résultat important. Toutes les bonnes pratiques d'un grand maître d'échecs pourraient être énoncées, car il pourrait toujours expliquer pourquoi il a joué tel ou tel coup. Le problème que nous pointons ici est plutôt celui de l'*usage* de ces règles. Consignées telles quelles, dans le désordre, elles seraient inutilisables.

102. Voir le débat à ce sujet dans les articles (Vera et Simon 1993a,b, 1994).

Réorganisées afin que leurs priorités et toutes leurs interdépendances soient prises en compte, elles résulteraient en un algorithme calibré par des gradients numériques qui ne nous dirait plus rien, et dont l'exécution serait elle-même problématique sans l'aide de machines. Cet algorithme n'aurait plus grand chose à voir avec ce que nous appelons le *savoir humain*. Les personnes, visiblement, procèdent autrement pour prendre des décisions éclairées au sein de disciplines complexes. Cela implique que les règles et les procédures par lesquelles ils communiquent ou discutent leur compétence jouent d'*autres rôles* que l'instruction algorithmique d'une séquence d'actions.

Nous arrivons ainsi au résultat important pour notre enquête que la démarche rationnelle de mise en procédure systématique à laquelle procède l'informatique semble, à bien des égards, *étrangère* à l'explicitation technique traditionnelle des savoirs pratiques. Cette explicitation qui se contente de « recettes de cuisine », qui est « intellectuellement légère » selon Simon[103] n'est pas un ersatz primitif de programmation, comme il le pensait, elle semble être *autre chose* que la programmation.

§ 72. Critiques et discussion

Ces résultats sont décevants. Nous sommes partis de l'idée attractive de Simon, que la programmation permettait d'exhiber la résolution de problème sans sa « pureté » d'acte de connaissance, et qu'elle pourrait ainsi permettre à toutes les professions ayant pour objet principal la résolution de problème (ingénierie, médecine, droit, etc.) de se constituer en sciences et même de s'enrichir les unes les autres de leurs méthodes.

Simon nous a par là permis d'entrevoir l'unité fondamentale de diverses formes de réflexion qu'on tiendrait naturellement pour distinctes : La conception (*design*) et la délibération, la démonstration, la planification, etc. – toutes ces activités ressortent d'une même activité de résolution de problème, qui a toujours une même structure fondamentale sous-jacente, l'*espace de problème*, lui-même constitué d'*états* et d'*opérations* transformant ces états. Cette structure fondamentale peut prendre des formes variées et riches, qui expliquent la diversité des langages de programmation. Cependant, il est toujours possible de présenter, *a minima*, toute résolution de problème comme une *recherche* (*search*) de solution dans l'espace de problème. L'idée d'une *systématicité* possible de la recherche apparaît ici, puisque

103. Voir notre citation plus haut § 63.

l'espace de problème peut être parcouru élément par élément à la recherche d'une solution.

Enfin, parmi les structures principales mises en évidence par Simon, celle d'*interface* entre un milieu interne et un milieu externe semble pouvoir rejoindre l'interprétation contractuelle de la programmation, dégagée au chapitre précédent. Toutes deux conçoivent le programme comme situé à l'interface entre une machine et une situation. Certes, c'est plutôt la spécification qui, dans notre schéma, occupe cette position, mais ce qu'entend Simon par *programme* est très proche de l'idée d'une spécification étendue à sa conception algorithmique, telle que nous l'avons exposée[104]. La *conception* est bien ce qui instaure le régime du *ought*, c'est-à-dire du caractère prescriptif de la spécification dans sa dimension contractuelle. De plus, pour Simon comme pour M. Jackson, ce qui importe est de bien représenter le problème comme transformation d'une situation à l'aide d'actions accessibles à la machine. Tous deux pointent donc bien vers des *théories des opérations* qui permettent d'expliciter le problème et l'espace des solutions possibles. Simon, on l'a vu, utilise plutôt l'expression *théorie de représentations*, mais comme une représentation n'est pour lui rien d'autre qu'un espace de problème, et que celui-ci se définit par le couplage essentiel entre état et action, c'est bien de ce que nous avons appelée de manière provisoire *théorie des opérations* qu'il est question. Simon nous permet d'ailleurs d'affiner cette idée, en montrant son caractère dual : *états* et *actions* (ou opérations) ne peuvent être envisagés séparément et pointent les uns vers les autres.

Cependant, toutes ces intuitions se heurtent aux deux difficultés que nous venons de mettre en avant dans cette section. D'une part, il ne semble pas possible d'assigner un domaine spécifique à une éventuelle *science de la résolution de problème* que serait la programmation. Le domaine des *pratiques professionnelles*, aussi large soit-il, est trop étroit, car la résolution de problème concerne également, et de manière évidente et même éminente, toutes les sciences et toutes les mathématiques. D'autre part, l'idée de départ, que la programmation serait la forme « achevée », « parfaite », ou « pure » – quel que soit le terme employé – de la résolution de problème nous semble finalement inacceptable. Il y a des résolutions de problème qui, au moins *phénoménologiquement*, ne passent pas par l'explicitation d'un espace de problème et la formulation de règles à son propos.

Sommes-nous pour autant revenus au point de départ de ce chapitre ? Il ne

104. Voir plus haut, § 50.

nous semble pas. D'une part, Simon, nous indique la manière par laquelle il faut aborder la notion de *résolution de problème*, et la richesse structurelle de ce champ de connaissance. En particulier, il nous montre combien l'heuristique de la fin et des moyens, à laquelle on réduit souvent la délibération pratique, est un cadre étroit pour représenter les méthodes et modes de réflexion de cette dernière.

D'autre part, les deux difficultés que nous avons rencontrées sont révélatrices de celle qu'il y a à situer la *résolution de problème* dans le champ de la connaissance. Premièrement, il s'agit d'une notion trop générique pour la circonscrire à un domaine particulier, fût-il aussi large que celui de la *connaissance pratique*, prise au sens du savoir de *ce qu'il faut faire*, par opposition à la *connaissance théorique*, prise au sens du savoir de *ce qui est*. La résolution de problème passe par-dessus cette distinction, aussi élémentaire qu'elle soit, que Simon exprime par le couple *is / ought*. Si on l'associe tout de même plutôt au savoir pratique, c'est qu'elle lui est essentielle (toute connaissance pratique est résolution de problème), tandis que la connaissance théorique, si elle comprend bien cette forme d'activité, ne s'y réduit pas : quoiqu'en dise Simon, on ne qualifie pas en général la construction de théories et la recherche théorique de *résolution de problème*.

Cependant (et c'est notre second point, relatif à la seconde difficulté), *au sein* de la résolution de problème, il semble qu'il y ait différentes manières génériques de procéder, l'une « intuitive » et « pratique », et l'autre plus méthodique, réglée et systématique, et qu'on aurait envie d'appeler technique.

Ces deux remarques mettent en évidence deux sens dans lesquels on peut entendre l'expression *savoir pratique*, que nous avons associée jusqu'ici, de manière assez vague, à l'idée de *résolution de problème*. Le premier sens de *savoir pratique* l'oppose à *savoir théorique* et caractérise l'*objet* spécifique de ce savoir (ce qu'il faut faire, *vs* ce qui est). Le second sens l'oppose à *savoir technique* et caractérise sa *modalité* spécifique (le savoir-faire, vs la méthode et les règles).

On voit ici combien les choses sont compliquées à exprimer avec les catégories que l'histoire de la philosophie nous a léguées. D'une part la programmation, en tant qu'elle est une résolution de problème, peut être mobilisée par des entreprises de connaissance aussi bien pratiques que théoriques. Cependant, elle se distingue radicalement de la modalité *pratique* de résolution de problème, et semble plutôt relever de sa modalité *technique*, qui emploie des méthodes et des règles.

Il y a là un nouveau fil directeur pour notre réflexion : pourrait-on « sauver » l'idée de Simon, en disant que la programmation est la forme achevée de la *ré-*

flexion technique, c'est-à-dire de la modalité spécifique par laquelle les professions et les sciences résolvent les problèmes, par l'énonciation de règles et de méthodes ? On a ici une hypothèse, en apparence bien plus acceptable que l'hypothèse computationnelle de l'esprit, qu'on pourrait appeler l'*hypothèse computationnelle de l'esprit technique*.

Cette hypothèse requiert de clarifier d'abord cette opposition « modale » entre *pratique* et *technique*, que nous n'avons qu'esquissée, et qui est assez visible dans l'usage de la langue.

4.4 Pratique(s) et technique(s)

Nous consacrons cette section à l'élucidation de l'opposition qui est faite couramment entre *pratique* et *technique* quant à la modalité d'une délibération. Le développement § 73 est consacré à l'analyse du terme *pratique*, dont les nuances commandent les points de vue différenciés de MacIntyre et de Bourdieu sur *les pratiques* humaines (au pluriel) (§ 74 et § 75). Les sens de *technique* sont ensuite examinés (§ 76). Ils révèlent les tensions sous-jacentes à la notion de *savoirs pratiques*. Le dernier développement § 77 revient sur l'*hypothèse computationnelle de l'esprit technique*, élaborée à la fin de la section précédente (c'est-à-dire l'idée qu'on pourrait assimiler la programmation à la forme achevée de la réflexion technique), et montre sa difficulté.

La partie terminologique de cette enquête s'appuie sur le *Trésor de la Langue Française*[105] et sur le *Petit Robert*[106].

§ 73. Pratique et pratiques : approche terminologique

Le radical *pratique*, qui apparaît en français vers 1350, a pour dérivations principales le verbe *pratiquer*, le substantif *pratique* et l'adjectif *pratique*. Nous examinons d'abord les sens assez restreints que prend sa forme verbale, puis les nombreuses connotations que leur ajoutent les autres formes.

Pratiquer est un verbe transitif. Que pratique-t-on donc ? D'une part, dit le dictionnaire, un art ou une science, mais également un sport ou une langue. Il y a là exercice régulier de domaines reconnus de compétence, où l'expertise est possible,

105. Dictionnaire *TLFi* en ligne, CNRS, Université de Lorraine, 1994, atilf.fr.
106. Dictionnaire *Le Robert*, Paris, 2017.

si bien qu'un champ disciplinaire peut émerger. Ces exercices se manifestent par des résultats tangibles et appréciés. La pratique est dans ce sens dès l'origine lié au savoir.

Il y a de nombreuses connotations à ce premier emploi du verbe. D'abord, il n'est pas synonyme d'agir, car il dénote d'emblée un agir régulier à travers le temps et qui a donc une forme reconnaissable, malgré la singularité possible des circonstances. On ne peut pratiquer un événement singulier, une action héroïque sur le champ de bataille par exemple. Mais on peut pratiquer la guerre.

Ensuite, il est plus général que l'exercice professionnel d'un métier. On peut pratiquer l'art de la ferronnerie sans être ferronnier, et l'astronomie sans être astronome. On utilisera d'ailleurs cette expression pour désigner spécifiquement l'amateur éclairé, capable de fréquenter les professionnels sans être du métier. On retrouve ici un vieux sens de *pratiquer*, celui de *fréquenter, avoir accointance avec* une personne ou un lieu.

En ce sens, le terme se dote d'une connotation d'*expérience*, qui s'oppose à l'accointance simplement théorique (au sens péjoratif) avec son objet. Pratiquer une langue, c'est ne pas en avoir une connaissance seulement scolaire.

Mais *pratiquer* peut également se colorer de la connotation opposée, lorsqu'il s'agit de *mettre en pratique* les enseignements d'une discipline, comme le montre l'expression *travaux pratiques* dans l'éducation scientifique. *Pratiquer la chimie* évoque par exemple l'idée d'un travail en laboratoire, qui complète plutôt qu'elle ne s'oppose à la dimension théorique de cette discipline. Ce sens d'*appliquer* se voit le plus clairement lorsque *pratiquer* est utilisé dans le domaine social et moral : *pratiquer une religion, la charité*, etc. Il devient alors synonyme d'*observer* des commandements et des règles.

Le verbe *pratiquer* a une autre grande catégorie de compléments d'objets, qui ne connotent pas des compétences ou des disciplines, mais des manières de faire. Les exemples les plus courants sont péjoratifs : *pratiquer l'usure, le chantage, etc.*. Mais il peut également viser plus largement des coutumes, comme dans *« cette communauté pratique la circoncision »*.

Ces deux grandes catégories de compléments du verbe *pratiquer* organisent l'usage du substantif *pratique*. Lorsqu'il s'agit d'une discipline, *pratique* s'emploie strictement au singulier, avec un déterminant défini. *« la pratique du football »* ne peut se mettre au pluriel ou à l'indéfini sans changer de sens. L'expression *« les pratiques du football »* désigne les usages et les méthodes particulières au champ

socio-économique de ce sport. Dans ce second sens, on parle également de *bonnes pratiques* pour désigner l'état de l'art d'une discipline.

Dans le premier sens, *pratique* reprend les connotations du verbe *pratiquer* en les accentuant, notamment par les expressions courantes *mise en pratique* (application) et *manque de pratique* (expérience). Même quand il s'agit d'application, comme dans *« la pratique d'une science »*, l'usage du substantif insiste davantage que ne le faisait le verbe sur la *différence positive* qu'ajoute cette pratique à la connaissance qu'on a de la discipline en question. On trouve donc ici l'idée que la pratique dépasse toujours la simple application des théories.

Surtout le concept (« la » pratique en général, sans complément d'objet) devient l'un des termes, avec « la » théorie, d'une opposition récurrente et transdisciplinaire, une sorte de *topos* aussi bien savant que populaire, qui oppose deux formes d'être dans le monde, l'une contemplative et l'autre agissante, ou encore deux formes de savoir, l'un issu de la réflexion pure ou appris dans les livres, l'autre de l'expérience de l'action et de la vie. Ce topos est donc traversé de multiples champs de valeurs qui peuvent donner l'avantage soit à l'un, soit à l'autre. On peut par exemple, comme le fait Nietzsche, glorifier la vie et rejeter le savoir sec et désincarné de la théorie comme d'une ruse de la vie elle-même, ou au contraire, condamner la *violence* que fait toute pratique aux choses dont elle s'occupe, au regard de l'attitude de désintéressement et de respect à l'égard des choses mêmes dont part toute entreprise théorique.

C'est dans l'adjectif *pratique* que se dénouent le plus clairement les fils de ces multiples connotations. Si on met à part le sens moral que lui accorde la seule tradition philosophique, notamment après Kant, *pratique* qualifie toutes choses qui ont trait à l'action, avec deux connotations majeures : la première, l'orientation vers l'utile, le concret et le matériel, notamment quand il qualifie des personnes ou leurs actions (*« quelqu'un qui a le sens pratique »* = quelqu'un qui a le sens de ses intérêts concrets), et la seconde, l'efficace, notamment quand il qualifie des outils ou des méthodes (*« un couteau pratique, un raccourci pratique, etc. »*).

§ 74. Les pratiques humaines selon MacIntyre

La petite enquête terminologique que nous avons menée nous permet de repérer les contrastes qui existent entre trois interprétations célèbres de la notion de *pratique*, celle d'Hannah Arendt, d'Alasdair MacIntyre et de Pierre Bourdieu.

Le premier point qui ressort de notre enquête est que la pratique a trait à l'action régulière. Elle a donc une certaine forme et généralité. En ce sens, elle s'oppose à *l'action*, notamment dans le sens qu'Hannah Arendt donne à ce dernier terme, lorsqu'elle insiste sur la singularité des événements que l'agir humain provoque, essentiellement imprévisible car révélatrice d'une singularité plus profonde encore, celle propre à chaque personne humaine. L'action, avec la parole qui lui est associée, fonde l'espace public où les personnes se rencontrent *en tant qu'*individus, où ils apprennent à se parler dans leur altérité essentielle[107]. En nommant *action* ce troisième domaine de la *vita activa* (avec ceux du *travail* et de l'*œuvre*) Hannah Arendt reconnaît implicitement l'impropriété de *pratique* dans la langue moderne pour signifier ce règne de la singularité, malgré son origine grecque et aristotélicienne à laquelle elle veut revenir : aussi est-elle contrainte d'utiliser la racine grecque telle quelle, *praxis*, quand elle veut s'y référer[108].

Le second point est que la *pratique* recèle une dualité sémantique, aux contours certes flous, néanmoins clairement perceptible dans l'opposition déjà citée entre *la pratique du football vs les pratiques du football*. On peut contraster les approches de MacIntyre et de Bourdieu sur la pratique en prenant pour fil directeur cette opposition.

Ce contraste est visible dans la décision que prend MacIntyre de *forcer* l'usage en qualifiant de *pratiques* (au pluriel) *tout ce dont il y a pratique* (au singulier), c'est-à-dire disciplines, arts, métiers, compétences. La définition qu'il en donne est célèbre :

> Par *pratique*, j'entends toute forme cohérente et complexe d'activité humaine, coopérative et établie socialement, par laquelle des biens internes à cette forme d'activité sont réalisés en essayant d'atteindre les normes d'excellence qui lui sont appropriées et qui la définissent partiellement, avec pour résultat que les pouvoirs humains pour atteindre l'excellence, et les conceptions humaines des fins et des biens impliqués, s'en trouvent systématiquement élargis. En ce sens, le jeu du morpion n'est pas un exemple de pratique, pas plus que tirer dans un ballon de football avec adresse ; mais le football l'est, et les échecs aussi. Empiler des briques n'est pas une pratique ; l'architecture l'est. Planter des navets n'est pas une pratique ; l'agriculture l'est. Le sont également les recherches de la physique, de la chimie et de la biologie, ainsi que le travail de l'historien, et encore la peinture et la musique. Dans les mondes antiques et médiévaux, la création et le maintien de communautés – familles, cités,

107. (Arendt [1958] 2012, p. 200).
108. Voir par exemple (ibid., p. 70, 79).

> nations – est considérée aussi comme une pratique dans le sens où je l'ai définie. Ainsi la portée des pratiques est large : arts, sciences, jeux, politique […], la formation et le maintien d'une vie de famille, tout cela tombe sous ce concept[109].

Cette définition force l'usage du français. Si nous acceptons sans difficulté l'expression « la pratique du football », nous sommes plus circonspects face à l'assertion que « le football est une pratique » comme le propose MacIntyre. Nous dirions plutôt *une occupation, un loisir, une activité, un métier*, etc. mais aucun de ces termes ne recouvre ce que vise le philosophe écossais. L'innovation lexicale de MacIntyre est peut-être un peu plus aisée à admettre en anglais, puisqu'il existe un sens du terme *practice* qui signifie directement l'activité professionnelle en exercice, qui peut aller jusqu'à inclure ce que nous appellerions en français le fond de commerce, comme dans les expressions « *he set up practice in London* » (il s'est installé (professionnellement) à Londres) ou encore « *he has a large medical practice* » (il a un gros cabinet médical). Cependant là encore MacIntyre tord l'usage anglais en l'élargissant bien au-delà de la stricte occupation professionnelle, puisqu'il y inclut certains loisirs, les sciences, la politique.

Le geste lexical de MacIntyre n'est pas anodin. Il ne s'agit pas seulement d'une synecdoque commode qui permet de passer de l'exercice d'une activité à l'activité elle-même. La visée de MacIntyre est ici évidemment éthique et anthropologique : en forgeant un concept *nouveau* de *pratique*, il cherche à réactiver l'idée ancienne de l'exercice *libéral* d'une discipline au sens ancien, c'est-à-dire libéré de la visée de biens externes (l'argent ou la renommée) au bénéfice de normes internes à la pratique, donc désintéressées, dotées d'une inscription sociale et d'une profondeur historique. Le concept qu'il propose est d'une certaine manière arbitraire. Il est moins large que l'activité tout court, qui peut être anodine, mais plus large que l'occupation professionnelle, ou même la *discipline*, terme qu'on réserve à des activités intellectuelles. L'exemple du jeu de morpion est significatif : s'il n'est pas une pratique, c'est qu'il est trop simple : il existe une procédure pour y gagner systématiquement ou faire match nul. Aussi ce jeu n'a plus de développement possible, et donc ne peut être un terrain à l'exercice et au dépassement de l'excellence humaine. Il est une simple technique. Pour une raison similaire, les compétences brutes à proprement parler – comme la compétence d'une langue, qui fait partie du champ sémantique de la pratique, semblent exclues de ce concept nouveau. Si on peut parler plus ou moins bien une langue, il n'est pas question que son

109. (MacIntyre [1981] 1985, p. 187).

simple usage devienne un terrain d'excellence – sinon à devenir un art, comme la rhétorique.

La possibilité de l'excellence qui caractérise une pratique est relative d'abord à des normes internes définies en son sein même. Elles peuvent donc évoluer avec elle. Ainsi la peinture n'a pas cessé de se redéfinir, mais également les sciences et les mathématiques. On pourrait citer à ce propos, et en guise d'illustration de cette conception de MacIntyre, la définition récursive des mathématiques proposée par le mathématicien Thurston :

> Les mathématiques sont le plus petit sujet satisfaisant les éléments suivants : 1) les mathématiques comprennent les entiers naturels et la géométrie plane et solide. 2) Les mathématiques sont ce que les mathématiciens étudient. 3) Les mathématiciens sont les humains qui font progresser la compréhension humaine des mathématiques. En d'autres termes, à mesure que les mathématiques progressent, nous les incorporons à notre réflexion. Au fur et à mesure que notre réflexion devient plus sophistiquée, nous générons de nouveaux concepts mathématiques et de nouvelles structures mathématiques : le sujet des mathématiques change pour refléter notre façon de penser[110].

Ces normes internes se traduisent en récompenses externes (honneurs, richesse, etc.) à mesure de la reconnaissance publique dont la discipline dispose, mais il est caractéristique d'une pratique qu'elle n'est pas guidée de manière essentielle par leur recherche, en quoi elle ne serait que mercenaire, selon le mot de Platon[111]. Cette *autonomie* des pratiques, au sens propre, conduit à leur conférer une sorte d'individualité historique trans-générationnelle, à en faire des sujets – on peut le constater à la syntaxe des développements de MacIntyre, où le terme apparaît principalement comme sujet grammatical, et souvent de verbes connotant l'activité et la vie[112].

MacIntyre contraste les pratiques aux techniques et aux institutions qui les soutiennent. Les institutions (clubs, laboratoires, syndicats, universités) organisent l'interface entre la pratique et l'espace public. Elles sont prioritairement concernées par la négociation des biens externes de la pratique ; elles sont nécessaires sur le long-terme pour assurer sa pérennité, mais elles peuvent être également un facteur de sa corruption, lorsque les jeux de pouvoir et d'argent prennent le pas

110. Cité par (Hacking 2014, p. 66).
111. *La République*, livre I, 346b.
112. (MacIntyre [1981] 1985, p. 189-193).

sur son développement autonome[113].

Les techniques sont un des moyens par lesquels l'apprentissage et la transmission d'une pratique sont réalisés. Mais une pratique est toujours plus qu'un ensemble de compétences techniques car celles-ci sont conditionnées par ses normes internes, dont on a vu qu'elles pouvaient évoluer. Si la peinture s'était sclérosée au point de se réduire à la technique de la perspective, jamais la peinture abstraite n'aurait pu être inventée. La transmission d'une pratique ne passe donc pas seulement par ses techniques, mais également par une tradition et des modèles d'excellence que leurs héritiers se donnent pour tâche de dépasser[114].

On voit par là que la codification du savoir d'une pratique ne peut jamais épuiser la pratique elle-même, pour au moins trois raisons. D'abord, sa codification en règles explicites est contingente à ses formes de transmission ; il se peut qu'elle passe plus aisément par l'apprentissage corporel de l'exercice. Par ailleurs, la transmission ne se réduit jamais à un savoir-faire, car elle porte également sur l'inculcation de normes et de modèles. Enfin, l'historicité même de ces normes et modèles (par opposition à l'immuabilité dogmatique) invite l'apprenti à rechercher leur dépassement – et c'est dans cette recherche que se définissent l'excellence et la maîtrise, autrement dit le savoir authentique de la pratique, *à un moment du temps*, car il récapitule tout son développement passé et ouvre vers ses possibilités futures.

§ 75. Le sens pratique (Bourdieu)

Il est intéressant de contraster la perspective éthique de MacIntyre avec celle, plus anthropologique et critique de Pierre Bourdieu. Pour le sociologue, le *sens pratique*, ou *habitus*[115], n'est jamais essentiellement arrimé à des disciplines particulières, ni réductible à la somme de leurs silos, mais au contraire permet leur émergence en tant que spécialisations du champ social.

Ce qui précède les pratiques au sens de MacIntyre, ce sont donc les pratiques dans le sens usuel que ce terme a en français lorsqu'il est employé au pluriel, c'est-à-dire les manières de faire et de se conduire. Elles sont les manifestations de l'*habitus* qui désigne la disposition à agir d'une manière déterminée face à des circonstances diverses. Cela ne veut pas pour autant dire que la conduite de l'individu est entiè-

113. (MacIntyre [1981] 1985, p. 194).
114. (ibid., p. 193-194).
115. L'équivalence des expressions est attestée par (Bourdieu 2015, p. 292).

rement déterminée par des facteurs externes et des lois sociales fixes. Au contraire, l'habitus est cette capacité justement à s'adapter à l'imprévu des circonstances, à se jouer des règles, à élaborer des stratégies au sein d'un champ social qui nous est donné. Cependant, cette adaptation elle-même se fait au sein d'un certain espace des possibles que l'habitus et la réalité externe, qui lui est ajustée, structurent. De nouvelles pratiques peuvent ainsi apparaître, mais elles s'inscrivent dans l'histoire des pratiques passées avec lesquelles elles ne peuvent jamais rompre totalement. L'habitus est inculqué à l'individu au travers de toute son expérience passée, qui est toujours sociale (la famille, l'école, le milieu professionnel), qu'elle passe par l'apprentissage explicite ou (le plus important) par l'observation silencieuse, ou le mimétisme inconscient du corps et de la parole. L'opposition du conscient et de l'inconscient n'a d'ailleurs pas d'importance ici, car la conscience qui accompagne une conduite, une stratégie ou une décision est toujours inscrite dans un fond impensé qui leur prête leur signification et leur possibilité même.

Dans ce cadre, les pratiques au sens de MacIntyre sont le résultat et le sédiment de stratégies passées, celles de groupes sociaux qui ont milité avec succès pour la délimitation d'un champ propre dont ils (les experts) établiraient les normes, les devoirs et les privilèges. C'est donc toujours ce même sens pratique qui est à l'œuvre à travers elles et toutes leurs prescriptions. Car le sens pratique élémentaire est celui de savoir reconnaître une norme *en tant que norme*, d'évaluer avec sagacité son pouvoir de coercition, de savoir quand et comment la contourner, quand et comment s'y conformer, de savoir la mobiliser à son avantage, etc. Les pratiques (au sens de MacIntyre) sont des pratiques (au sens de Bourdieu) que certains groupes ont réussi à imposer comme normes. Elles correspondent à ce que sociologue français appelle plutôt *champ*, c'est-à-dire l'espace objectif dans lequel ces pratiques peuvent se développer dans l'autonomie de leurs normes propres.

Bourdieu récuse donc radicalement ce qu'il appellerait peut-être l'irénisme sous-jacent à la notion de biens internes promue par MacIntyre. Les praticiens d'une discipline ne cherchent pas à se conformer ou à remettre en cause les normes de celle-ci dans un désintéressement sincère. Certes ces biens internes ne sont pas nécessairement monnayables en biens externes, mais ils constituent tout de même un capital symbolique qui est un enjeu de pouvoir interne au champ, qui est une structure de lutte et de domination sociale. L'évolution des normes ne suit donc pas une sorte de nécessité interne purement idéale, mais résulte d'un affrontement entre acteurs concernant la définition, et donc le contrôle du champ

tout entier – cela notamment dans les champs semi-structurés que sont les arts et les lettres.

Au-delà de cette critique concernant la « vraie nature » des soi-disant biens internes d'une pratique, la conception de Bourdieu rejoint singulièrement celle de MacIntyre.

D'abord, il est tout autant critique de l'idée d'une réduction possible des pratiques – quelles qu'elles soient, disciplinaires ou non – à leur codification en règles. Codifier et expliciter une pratique n'est pas un geste anodin : il signifie que celle-ci (ou plutôt l'acteur qui la codifie) est en danger : son apprentissage ne va plus de soi, d'autres normes sont en train d'apparaître, elle est contestée, etc. Elle participe donc peut-être à la consolidation défensive d'un champ, ou bien à une lutte stratégique entre deux groupes sociaux au sein de ce champ, – mais il en résulte que cette explicitation est toujours partielle, sinon partiale. Elle ne dit, en négatif, que ce qui ne va plus de soi, mais laisse sous silence toutes les structures essentielles qui continuent de réguler l'habitus et l'espace général dans lequel la règle a tout simplement un sens.

Ensuite, on retrouve, comme chez MacIntyre, la notion d'excellence comme marque réelle du savoir pratique authentique. Commentant la position de Durkheim sur l'art, qui peut être éclairé par la réflexion, mais qui n'en est jamais l'élement essentiel, Bourdieu écrit :

> On ne saurait mieux dire que la docte ignorance qui est au principe des stratégies quotidiennes ne doit pas s'exprimer dans le lexique de la règle, mais dans celui qu'emploient toutes les sociétés pour décrire l'excellence, c'est-à-dire la manière et les manières de l'homme accompli : cet « art sans art » […] ne se réalise jamais aussi complètement que dans les occasions socialement aménagées où, comme dans les joutes d'honneur, le jeu avec la règle fait partie de la règle du jeu. Si cette incarnation particulièrement réussie de la manière particulière d'être homme que reconnaît un groupe déterminé est presque toujours définie comme indéfinissable, parce que toute mise en formule la ravalerait au rang de simple procédé ou de truc mécanique, c'est que la virtuosité n'a que faire de la règle, garde-fou ou pense-bête, à peine capable de suppléer aux manques de l'habitus ; si elle se reconnaît à son « naturel », c'est qu'elle instaure cette maîtrise magique du corps propre qui, comme l'observe Hegel, caractérise la dextérité ou, dans la langue de l'honneur kabyle, la « grâce » du *sarr* […][116]

116. (Bourdieu [1972] 2000, p. 304).

La règle est donc « demi-savante[117] » (au sens du *demi-habile* pascalien) non seulement parce qu'elle est un entre-deux pataud entre l'ignorance complète d'une pratique et sa « docte ignorance » qui est en réalité sa vraie maîtrise, mais aussi parce que, au niveau épistémologique, elle est une transmutation incomplète du savoir pratique en savoir théorique :

> On comprend mieux pourquoi cette production demi savante qu'est la règle constitue l'obstacle par excellence à la construction d'une théorie adéquate de la pratique : en occupant faussement la place des deux notions fondamentales, la matrice théorique et la matrice pratique, le modèle théorique et le sens pratique, elle interdit de poser la question de leur relation[118].

Au-delà donc de leurs différences, MacIntyre et Bourdieu s'accordent sur cette propriété essentielle du savoir pratique, de ne pouvoir jamais entièrement, ni même essentiellement, se codifier en règles. Celles-ci apparaissent comme un de ses modes possibles d'expression, partiel et provisoire, et nullement comme un objet de travail essentiel à la réflexivité dont certaines pratiques ont besoin pour se développer en tant que telles. Sans qu'ils nient cette possibilité, ni Bourdieu ni MacIntyre ne s'intéressent à la fécondité possible de l'écriture même des règles dans la constitution d'une pratique ni, *a fortiori* de la simple combinatoire des règles en un système, qu'est un programme.

Or c'est cette possibilité de la programmation qu'il nous intéresse ici d'établir et de comprendre, notamment dans son rapport au savoir pratique qui, lui, est irréductible aux règles. Cela nous amène à étudier l'usage du terme *technique*, qui lui est directement associé.

§ 76. Technique et techniques : approche terminologique

Le terme *technique* apparaît en français dans sa forme adjectivale au siècle des Lumières, dans le cadre de la discussion sur les arts et métiers. Il faut attendre un siècle environ d'une histoire tourmentée retracée par Camolezi (2022), pour que le substantif *la technique* soit attesté par les dictionnaires, à commencer par le Littré qui en donne la définition canonique, reprise jusqu'à aujourd'hui : « L'ensemble des procédés d'un art, d'une fabrication[119]. » Nous reviendrons plus loin sur l'histoire de cette genèse qui s'ancre dans la problématique des arts et métiers

117. (ibid., p. 308).
118. (Bourdieu 1980, p. 176).
119. Cité par (Camolezi 2022, p. 72).

et de la *réduction en art* aux XVI^e^ et XVII^e^ siècles[120].

Aujourd'hui, les dictionnaires[121] distinguent deux nuances à cette définition. L'une, plus large, porte sur les procédés de toute « activité » (notamment d'un art), « permettant d'obtenir un résultat concret ». Dans le parler courant (comme par exemple *« il a une technique pour se défiler »*), le terme peut même devenir synonyme de *procédé* en général, indépendamment de la référence à tout art ou discipline.

L'autre nuance, plus étroite, porte sur les procédés « méthodiques, fondés sur des connaissances scientifiques, employés à la production ». Une double circonscription se fait ici, d'abord quant à l'inspiration scientifique des techniques, ensuite quant à leur application au domaine strict de la production. Ce second sens n'est donc pas loin de l'une des acceptions courantes de la *technologie*, comme « science appliquée à l'industrie ». Cette seconde nuance fait apparaître l'ambiguïté du terme *procédé*, que le dictionnaire décrit de manière assez large comme « façon d'agir, méthode employée pour parvenir à un résultat » ou encore « forme de déroulement d'un processus ». *Procédé* occupe ainsi une place ambiguë entre une procédure, prescriptive de l'action, et un processus objectif. Dans l'industrie, le procédé est entendu d'abord en ce second sens, quand on parle d'un procédé d'injection ou d'un procédé chimique. Mais l'autre sens, qui prédomine dans la première nuance de *technique*, y est également possible. D'une manière générale, donc, une technique peut prescrire une conduite à l'agent (sens « subjectif ») ou décrire le comportement des choses (sens « objectif »). On voit comment un sens peut se transmuer aisément en l'autre, puisqu'il suffit de changer de point de vue : ce qui est une technique subjective pour l'ouvrier (une procédure) est objective pour l'ingénieur qui l'observe agir (un processus).

Il faut également noter deux emplois au singulier déterminé (« la » technique) qui viennent chacun accentuer ces deux nuances. Dans le sens du rétrécissement, *la technique* désigne « l'ensemble des procédés, scientifiquement mis au point, qui sont employés à l'investigation et à la transformation de la nature[122]. » C'est la technique telle qu'on en parle notamment en philosophie contemporaine, et qui n'est pas loin du concept actuel de *technoscience*. Hannah Arendt, en identifiant la technique au second pan de la *vita activa*, qu'elle appelle l'*œuvre*, insiste sur

120. Voir plus loin, section 7.1.
121. Le *Petit Robert* et le *Trésor de la Langue Française*.
122. *Le Petit Robert*.

sa seule dimension productive (qu'elle soit inspirée scientifiquement ou non). Elle reconduit par là la distinction forte entre *praxis* et *poiesis* instituée par Aristote. Selon Arendt, l'œuvre recouvre les agissements des personnes humaines visant à aménager le monde pour le rendre habitable à travers les générations, tandis que l'action recouvre ceux par lesquelles les personnes se mettent en rapport les uns avec les autres. Nous n'avons pas la place de critiquer ici cette dichotomie, mais nous ne la suivons pas dans ce travail. On verra que le texte aristotélicien lui-même aménage une assez grande circulation entre les deux domaines, notamment au travers de l'idée d'*habileté* qui participe aussi bien de la technique que de la pratique[123]. Surtout, elle n'est pas conforme à l'usage courant qui articule aujourd'hui *pratique* et *technique* plutôt qu'il ne les oppose.

Cette articulation se voit dans l'élargissement de la portée du concept de *technique* dans le sens où on parle de *la technique d'une personne* pour désigner son *savoir-faire* ou son *habileté* à pratiquer une activité, autrement dit de ses compétences personnelles et non des procédés qu'elle emploie. Il n'est pas évident que *« la technique d'un pianiste »* puisse être entièrement décrite par *« des techniques du piano »*. Cependant, l'emploi spécifique de ce terme, au lieu de *savoir-faire* ou *habileté* peut connoter l'idée d'une description *possible* de cette compétence en procédés. Si la technique d'un pianiste peut être virtuose, c'est qu'elle se situe au sommet d'une gradation *continue* de difficultés, donc potentiellement accessible à d'autres personnes. D'ailleurs, de nombreuses méthodes pédagogiques, notamment dans les arts, portent le nom de maîtres ayant ainsi réussi à formaliser et organiser leur propre compétence, comme la méthode Hanon au piano.

Il importe dans une technique que les conditions de sa pratique réussie soient explicitées, par exemple que l'apprenti ait maîtrisé tel ou tel geste, voire que son corps, comme sa main, ait telle ou telle caractéristique, avant de pouvoir appliquer avec succès la règle prescrite. Une technique promet des résultats à tout agent appliquant une règle *sous certaines conditions* explicites ou explicitables. Ainsi, pour revenir au monde industriel, et établir la *réciproque* de l'équivalence entre sens objectif et subjectif de *technique*, un procédé objectif (telle cadence de production par exemple sur une chaîne de montage) peut être déclinée en procédures subjectives pour les ouvriers *du moment qu*'on a déterminé en termes généraux quelles étaient les compétences attendues de ces derniers, qui rendent leur action effective. Une technique suppose toujours, quelle que soit sa difficulté, un fond de *normalisation*

123. Voir plus loin, § 96.

de l'action.

Les emplois de l'adjectif *technique* ajoutent un sens complémentaire aux précédents, qui insiste sur la *spécialisation* de l'activité ou de la connaissance en question : « *un ouvrage, un vocabulaire, une expression techniques* » ne sont compréhensibles que par des personnes spécialisées, versées dans la discipline.

§ 77. La programmation et les techniques

Au terme de cette enquête, trois choses apparaissent :

1) Sur l'ensemble du spectre de leurs usages, *pratique* et *technique* portent toujours sur des actions dotées d'une certaine régularité et d'une certaine publicité. L'action singulière et entièrement privée n'est pas de leur ressort.

2) On trouve dans les deux termes une même tension entre une acception circonscrite à certaines activités régulières, bien idenfiées socialement, et une acception plus large qui vise plus généralement des compétences ou des manières de faire. Sur cette échelle, *technique* s'emploie bien plus souvent que *pratique* dans le contexte étroit d'une discipline spéciale, puisqu'il peut aller jusqu'à signifier *le fait même* de la spécialisation. Cependant on voit qu'il peut aussi aller vers des usages très génériques, notamment dans des locutions populaires.

3) Les deux termes sont également traversés par une seconde tension, entre l'expérience personnelle de l'action, intransmissible, et sa possible description externe, normalisable et transmissible. Nous avons vu cette tension dans l'opposition entre *la pratique* d'une discipline, toujours individuelle, et *les pratiques* qui désignent les manières communes de faire. Elle est également présente, dans une moindre mesure, dans l'emploi de *technique* comme savoir-faire (personnel), qui contraste avec son sens principal comme *l'ensemble de procédés* qui forment le fond transmissible de la pratique d'une discipline. Dans le terme *technique*, ce contraste va jusqu'à désigner un sens objectif, qui ne recouvre pas seulement les conduites prescrites aux agents, mais l'agencement même des choses de façon à en déterminer le comportement.

Ce sont donc ces deux tensions, d'une part entre le caractère général *ou bien* spécialisé (disciplinaire) de l'action régulière, et d'autre part entre son caractère privé et subjectif *ou bien* son caractère public et objectif, qu'il est essentiel d'interroger au-delà des nuances de l'emploi des termes *pratique* et *technique*.

En nous autorisant à les projeter chacun vers leurs polarités principales, le premier terme désigne donc d'abord le *sens pratique*, cette faculté *générale* de

s'adapter au commerce du monde, d'y gagner de l'expérience, d'y appliquer son savoir, faculté qui se *spécialise* localement en *pratiques particulières* (professions, arts, sciences, jeux, etc.) *transmissibles par des techniques* et structurées par des institutions. Les techniques sont donc de l'ordre du savoir, puisqu'elles sont des procédés ou des méthodes qui, afin d'être transmissibles, doivent pouvoir être représentées. Les techniques se décrivent et s'enseignent. Elles ont bien trait à ces ensembles de règles, partiels et provisoires, auxquels le savoir pratique ne saurait se réduire. Elles peuvent déterminer aussi bien la conduite des personnes que le comportement des choses.

Il est aisé à partir de là de décliner une série d'oppositions ou plutôt d'articulations associées à ces deux termes, qui remontent à leur origine grecque : la pratique est inscrite dans les affaires humaines, tandis que la technique vise le « gouvernement des choses » (une technique portant sur de l'humain, comme la publicité, le traite comme un objet normalisable) ; la pratique a une histoire (l'expérience personnelle, la tradition collective) tandis que la technique vise l'effectivité toujours actuelle des méthodes ; la pratique s'adapte aux circonstances, la technique ne traite que de cas de figure réguliers.

Il est pertinent de parler d'articulation plutôt que d'opposition, car pratique et technique apparaissent, en ce sens que nous leur donnons, comme les deux termes d'un seul mouvement d'extériorisation et de structuration sociale de l'action régulière, et qui est bien sûr réciproque, puisque le sens pratique est largement, sinon tout entier fait de l'intériorisation et de l'habituation aux structures externes et publiques du monde. On parle bien de la description technique d'une pratique (extériorisation) et, en retour, de la mise en pratique d'une technique (intériorisation).

Où situer la programmation dans cette perspective ? Il peut sembler évident, à première vue, qu'elle se situe du côté des techniques, puisqu'elle a trait à l'explicitation de règles pour l'action. La polyvalence des ordinateurs à travers de si nombreuses disciplines, que nous avons posée comme problématique au début de ce travail, s'expliquerait simplement par ce que les programmes *codent* leurs techniques. Ce travail de codage permettraient à ces règles d'apparaître dans toute leur clarté et leur forme achevée, ce qui justifierait l'expression d'*hypothèse computationnelle de l'esprit technique* par laquelle nous avons proposé de caractériser la programmation à la fin de la section précédente.

Cependant deux difficultés majeures apparaissent ici.

D'une part, si les techniques sont spéciales à chaque pratique ou à chaque discipline, la programmation, qui permet de les coder toutes, *n'en est pas une elle-même*. Consistant entièrement dans l'élaboration de systèmes de règles, elle est une manière de *rendre technique* une pratique, et donc ne peut être une technique elle-même, un simple procédé.

Certes, si la programmation ne consistait qu'en la traduction en code informatique de techniques déjà parfaitement explicitées, on pourrait tenter de soutenir une telle thèse. Mais nous avons vu au chapitre précédent combien un projet de programmation était en général éloigné d'un tel cas de figure.

D'autre part, l'explicitation des rapports entre modalités *pratiques* et *techniques* du savoir montrent qu'elles sont étroitement associées. Aussi, l'idée d'immenses livres de procédures[124] semble hostile, non seulement au sens pratique des agents humains, mais également aux techniques dans lesquelles il le codifie.

Ce sont ces difficultés que nous allons aborder au chapitre suivant.

124. Voir plus haut, § 71.

Chapitre 5

Raison pratique et logique industrielle

§ 78. Introduction

Nous avons appelé réflexion technique la réflexion qui vise à expliciter le savoir propre à une pratique donnée de manière à ce qu'elle puisse être examinée et transmise. Cette réflexion est elle-même critique du savoir-faire immédiat d'une pratique, même si la manière dont elle s'en affranchit, notamment dans la technique moderne, ne nous apparaît pas encore clairement[1]. On rappelle que *technique* ne se limite pas ici au domaine de la production mais qu'il inclut toutes les professions, comme la médecine, le droit, ou l'administration.

La thèse des *Sciences de l'Artificiel* de Simon, une fois dissociée de l'hypothèse computationnelle de l'esprit, nous a semblé attractive. Elle implique en effet que la réflexion technique, parvenue à maturité, est un travail de *conception*, dont la forme générale est la programmation, telle que nous l'avons décrite au chapitre 3. Cette thèse expliquerait parfaitement la polyvalence des ordinateurs à travers de nombreuses pratiques et entrerait en résonance avec l'opinion courante – on la trouve par exemple chez Heidegger – que l'informatique est une forme d'aboutissement de la technique.

Cependant, affirmer que la programmation est la forme générale de toutes les formes de rationalité discursive pratique se heurte au caractère proprement monstrueux à laquelle pourrait aboutir, dans de nombreuses disciplines, l'expression systématique d'un enchevêtrement de règles dont la priorité serait relative aux situations rencontrées, comme l'a montré l'exemple des jeux de stratégie. À cet égard, le droit semble faire figure d'exception. La capacité qu'ont les législateurs à

1. Voir plus haut, § 77.

garder à peu près en ordre leur corpus de règles, ainsi que le langage élémentaire dont ils se servent, et cela malgré les changements du monde et le poids toujours croissant de la jurisprudence, semble n'avoir d'équivalent dans aucune autre pratique humaine, y compris l'ingénierie et la médecine. Il n'est pas lieu dans ce travail d'explorer un tel fait, cependant on peut avancer que cette situation exceptionnelle pourrait tenir justement au fait que le droit s'occupe spécifiquement des « pactes et des conventions » entre les hommes, selon le mot de Hobbes[2]. L'exigence d'un ordre réglé entre toutes les prescriptions du droit découlerait directement de la fonction des conventions, d'être un lieu d'intelligibilité où peuvent s'accorder les hommes.

Cependant il suffit pour notre enquête, qui cherche seulement à caractériser la forme spécifique de connaissance qu'est la programmation, de comprendre de quelle manière elle se distingue des autres formes de rationalité procédurale qu'on peut observer dans les autres pratiques humaines, et comment, *malgré* cette différence, elle est capable de les transformer *tout de même*.

Pour ce faire, nous proposons de restreindre notre étude aux rapports qu'entretient la programmation avec un ensemble de pratiques professionnelles qu'on pourrait regrouper sous le titre général d'*opérations en milieu technique*. Avant de justifier ce choix, il faut expliciter ce qu'il recouvre.

Ce qu'on entend par *opérations* se dit parfois encore *exploitation* dans certains secteurs industriels, mais ce vocable tend à disparaître au profit du premier, importé de l'anglais mais parfaitement correct en français, et plus général. Ainsi, l'exploitation d'une usine, la logistique, la gestion d'un centre d'appel, la conduite d'un grand navire, la maintenance d'infrastructures ou d'un parc de machines, les activités d'une tour de contrôle, etc. tout cela est couramment qualifié d'*opérations* aujourd'hui, par opposition d'une part aux activités commerciales et administratives d'une entreprise (même si, par exemple, la gestion d'un *back-office* comptable peut aussi être qualifié, par extension, d'*opérations*) et d'autre part aux activités de projets, notamment recherche, développement et construction. Ainsi, on l'a vu[3], un département informatique est lui-même doté d'activités d'*opérations*, traditionnellement séparées de celles de développement. Le point commun de toutes ces activités est d'avoir une très grande récurrence, d'avoir des résultats bien définis et mesurables (par exemple la production pour une usine, ou le nombre d'appels

2. *Leviathan*, introduction.
3. Voir plus haut, § 21.

traités pour un service client) à périodes fixes (quotidienne, mensuelle, annuelle) qu'on peut mettre en regard de facteurs de production (quantité de travail, matières, etc.) si bien que l'effectivité et la productivité peuvent en être mesurées. Par l'expression *en milieu technique* nous voulons dire que ces activités se déroulent dans un cadre préparé pour maximiser leur effectivité et leur efficacité, c'est-à-dire normalisé et procéduralisé.

Le choix de ce domaine de pratiques est intéressant pour réfléchir à la spécificité de la programmation car c'est là, *a priori*, qu'on s'attendrait à trouver la plus grande proximité entre les procédures destinées aux opérateurs humains et les procédures informatiques. Ne dit-on pas que les opérateurs, en milieu industriel, doivent se comporter « comme des robots », et n'a-t-on pas tous à l'esprit les images frappantes de Charlot dans les *Temps modernes*, caricaturant et dénonçant le « management scientifique » de Frederick Taylor ? C'est donc dans ces pratiques professionnelles qu'il est *a priori* le mieux possible de retracer la proximité et l'écart entre ces deux formes de procédures.

L'usine moderne est le milieu technique par excellence. Elle est le lieu de la machinerie et de l'automatisation où l'opérateur, comme le notaient déjà Ure, Babbage et Marx, est *asservi* à la machine : les procédures qu'on lui demande de suivre sont les reliquats de ce qui n'a pu être encore automatisé, pour des raisons techniques ou économiques. Or l'avènement de l'industrie moderne, mécanisée, est également le moment de l'invention conceptuelle de l'ordinateur, par Charles Babbage qui associa étroitement les deux idées.

Avant d'explorer le sens de cette proximité historique et conceptuelle entre procédures informatiques et industrielles, nous commençons par observer le rôle des procédures dans des milieux techniques *non industriels*, où la *mécanisation* ne joue pas un rôle central ou directeur. Deux études d'ethnologie du travail, portant l'une sur les opérations de l'équipage d'un navire militaire, et l'autre sur la maintenance d'un parc de machines bureautiques, concluent toutes deux que les règles et les procédures écrites jouent un rôle très spécifique et exceptionnel dans l'organisation du travail humain. Si l'action collective parvient à l'effectivité normée, c'est d'abord grâce à la mise en place d'*écologies cognitives* complexes, mobilisant des ressources matérielles (agencement de l'espace, aides visuelles et sonores, outils, instruments, etc.) et sociales (apprentissage, compagnonnage, hiérarchie, etc.) qui contraignent implicitement le comportement des agents. Au sein de ces écologies, les règles et les procédures fonctionnent comme des points de référence

plutôt que comme des normes actives, et se concentrent sur la spécification, par exception, de points particuliers importants qui *complètent* les contraintes déjà imposées par l'écologie cognitive plutôt qu'elles ne les décrivent. Pour reprendre une image du philosophe de la connaissance Andy Clark[4], elles ne sont que la pointe visible d'une montagne cognitive sous-marine, ancrée dans les profondeurs sociales et corporelles du comportement humain (section 5.1).

Il y a donc une opposition quasi-frontale entre la logique des procédures réglant les conduites humaines et celle des procédures industrielles et informatiques. Pour les premières l'ambiguïté est une ressource qui permet de s'adapter aux situations nouvelles, tandis que pour les secondes, elle signifie un manque de maîtrise et de planification. Les unes et les autres se distinguent donc par un arbitrage différent entre la *durée* requise pour l'adaptation et la *garantie* du résultat. La procédure humaine peut être ambiguë car elle est fondée sur une écologie cognitive sous-jacente qui permet aux opérateurs humains de s'adapter aisément, par leur « sens pratique » aux situations nouvelles. Cependant elle ne garantit pas au maître d'ouvrage que cette adaptation sera conforme à ses attentes, ni même homogène d'un opérateur à l'autre, ce qui devient un frein majeur pour le passage à la grande échelle. L'industrialisation au contraire est tout entière tournée vers l'expurgation de tout facteur d'imprévu et d'initiative incontrôlée hors du processus de travail. Cela requiert un long temps d'analyse de l'espace des possibles. C'est la mobilisation *systématique* de la réflexion technique qui la rend industrielle et qui l'amène par là à placer la machine au centre du processus de travail, et la personne à sa périphérie (section 5.2).

Dans ce cadre, l'informatique apparaît comme la continuation directe et l'achèvement de la logique de transformation industrielle à l'œuvre depuis (au moins) le XVIII^e^ siècle, qui vise à objectiver le processus de travail et le rendre indépendant des contraintes subjectives humaines, qu'elles soient physiques ou cognitives (section 5.3). Observer l'informatique nous permet d'identifier de manière plus générale que ne pouvaient le faire les observateurs de la première révolution industrielle les avantages procurés par la mécanisation du point de vue de l'effectivité, dont l'automatisation n'est qu'un seul aspect (section 5.4).

L'informatique et l'industrie ont donc en commun une approche *systématique* de la réflexion technique, qui exige d'observer la pratique humaine comme un objet, c'est-à-dire théoriquement. C'est sur la modalité distinctement *systématique* et

4. (CLARK 2010, p. 75).

théorique (par opposition aux modalités *pratique* et *technique* mises en évidence à la fin du chapitre précédent) de la résolution de problème propre à la programmation et à l'organisation industrielle que nous concluons, de manière problématique, ce chapitre (section 5.5).

5.1 Le rôle des procédures au sein des pratiques humaines

Cette section est destinée à montrer que les procédures, telles qu'elles apparaissent au sein des techniques humaines, ne sont pas des ersatz de programmes. Elles jouent un tout autre rôle qu'esquisser une possible automatisation de la pratique. Dans le développement § 79, nous suivons une première étude d'ethnologie du travail en milieu technique. L'ethnologue HUTCHINS (1995), étudiant les opérations sur un navire de guerre, montre que les livres de procédure ne tentent pas de décrire le savoir pratique des marins pas à pas, mais viennent s'insérer dans une *écologie cognitive* matérielle et sociale qui forme le soubassement de la pratique. Les procédures sont des artefacts cognitifs coûteux pour l'esprit humain, qui servent justement à diriger son attention sur des points délicats (§ 80). Dans un dernier développement § 81, nous nous intéressons à la manière dont Hutchins interpréta initialement ses propres résultats en déplaçant seulement la métaphore computationnelle de l'individu au groupe organisé : pour lui, la rationalité pratique se réalise d'abord au niveau collectif, et c'est donc l'organisation qui « calcule ». Nous montrons les difficultés internes à cette perspective, en la comparant à des vues similaires qui se développent à partir des années 1990, et qui tentent d'expliquer l'effectivité des ordinateurs dans notre monde par leur rôle de facilitation de l'action organisée.

§ 79. La raison pratique : une affaire d'organisation ?

Diriger un grand navire est une activité rationnelle qui exige organisation et méthode et, dans un contexte militaire, les tactiques et les manœuvres sont consignées explicitement dans des livres de procédure auxquels l'équipage est tenu de se conformer. Cette activité semble donc pouvoir être décrite comme un système de règles. Hutchins va même jusqu'à qualifier le fonctionnement du navire comme un

système de production[*5], un concept qu'il emprunte donc à l'informatique, et il affirme que l'interaction globale de l'équipage, de leurs instruments et de la situation peut être décrite comme un calcul, si on prend ce terme dans son sens simonien, c'est-à-dire comme une transformation de représentations selon des règles. Il s'agit là d'une « métaphore » que Hutchins fait sienne :

> Bien qu'il ne soit pas habituel de parler des propriétés informatiques[6] d'institutions sociales, l'équipe de navigation peut être considérée comme une machine informatique. Dans cette section, j'explore cette métaphore en examinant les façons dont certains aspects du comportement du système peuvent être interprétés dans un cadre informatique[7].

L'ouvrage de Hutchins, qui date de 1995, est intéressant pour l'histoire des sciences cognitives. Déjà critique de l'hypothèse computationnelle de l'esprit, comme on va le voir, il maintient cependant une autre grande thèse simonienne, que nous n'avons pu développer, à savoir que les organisations (rationnelles) sont également des systèmes de traitement de l'information, *i.e.* calculent comme l'esprit humain et l'ordinateur. Cependant, les observations de Hutchins sapent de l'intérieur l'ensemble de la métaphore informatique et en cela portent le ferment des sciences cognitives « 4E »[8], qui apparaissent à ce moment et qui citeront fréquemment son ouvrage.

Cette tension est visible dans le jugement que porte Hutchins sur les livres de procédure :

> Les procédures écrites ne sont pas utilisées par les membres de l'équipe de navigation comme des ressources structurantes lors de l'exécution de la tâche, ni ne décrivent les tâches réellement effectuées. De plus, si un système était vraiment conçu pour fonctionner comme spécifié littéralement dans les procédures, il ne fonctionnerait pas. Néanmoins, les procédures normatives sont un bon point de départ et fournissent un cadre stable dans lequel les propriétés du système peuvent être décrites[9].

Comment est-il possible qu'une machine informatique fonctionne sans l'usage de procédures, et que leur usage puisse au contraire conduire à son échec ? C'est à réconcilier ces deux assertions en apparence opposées que nous allons nous attacher

5. (Hutchins 1995, p. 199).
6. Nous traduisons dans tout ce passage *computational* par *informatique*, mais on aurait pu également utiliser le terme *calculatoire*.
7. (Hutchins 1995, p. 185). Voir également le chapitre conclusif, p. 353, où il revient avec force sur cette thèse.
8. Voir plus haut, § 17.
9. (Hutchins 1995, p. 178).

ici.

Un concept central pour Hutchins est celui d'*écologie cognitive*, que la navigation illustre bien. Naviguer implique de se représenter la situation de son navire et sa destination, cependant il existe plusieurs possibilités pour ce faire. Les situer sur l'espace fixe d'une carte bi-dimensionnelle n'en est qu'une parmi d'autres. Pour les micronésiens par exemple, c'est la position du navire relativement aux étoiles qui est directement signifiante, ainsi que d'autres repères spatiaux, comme les teintes de la mer qui indiquent le relief sous-marin. Le navigateur micronésien, une fois son cap fixé, estime sa distance à son point d'arrivée en associant celui-ci à la course d'une étoile donnée[10]. Pour les marins occidentaux au contraire, naviguer est une activité médiatisée par un ensemble d'instruments et d'artefacts symboliques complexes. Par exemple, afin de connaître la position du navire et le placer sur une carte, il faut établir des relevés grâce à une boussole et des repères visibles depuis le navire (un phare, un promontoire) qu'on sait localiser sur une carte. Pour que cette technique soit précise, une transformation complexe de représentations doit avoir lieu entre plusieurs espaces, l'espace réel d'observation du repère à l'horizon grâce à une lunette tournante, l'espace du registre sur lequel l'angle de vue est consigné, puis celui du compas sur lequel il est reporté, et enfin celui la carte marine où une ligne de position passant par le repère peut être tracée[11]. Ainsi la position est calculée sans pour autant qu'aucune opération de calcul ait été faite par aucun marin :

> Ces outils transforment la tâche que la personne doit accomplir en la représentant dans un domaine où la réponse ou le chemin vers la solution est apparent. L'existence d'une telle variété d'outils et de techniques spécialisés témoigne de l'importance de l'élaboration culturelle visant à éviter le raisonnement algébrique et l'arithmétique. […] Ces outils permettent aux personnes de réaliser les tâches requises en faisant le genre de choses pour lesquelles elles sont douées : reconnaître des schémas, modéliser des dynamiques simples du monde et manipuler des objets dans l'environnement[12].

Cette écologie cognitive n'est pas seulement composée d'instruments et d'artefacts symboliques. Les éléments naturels observables sont également mobilisés (repères visuels, étoiles), bien que dans une très moindre mesure relativement au navigateur micronésien. Mais surtout, Hutchins insiste sur le caractère distribué et coopératif

10. (ibid., p. 92).
11. (ibid., p. 119-123).
12. (ibid., p. 155).

de cette écologie cognitive. Chaque processus est conduit par plusieurs membres de l'équipage communiquant entre eux, ce qui permet à chacun de se focaliser sur des tâches simples. Cela contribue de manière essentielle à créer un système de contraintes cognitives dont la satisfaction se trouve résoudre le problème :

> La composante humaine du système [agit] comme un tissu de coordination malléable et adaptable, dont le travail est de veiller à ce que les activités appropriées soient menées. Dans leur communication et dans leurs actions communes, les membres de l'équipe de navigation se superposent au réseau des médias calculatoires matériels. Ils fournissent le tissu de connexion qui déplace l'état de représentation à travers les outils[13].

La caractéristique essentielle de ce système est d'être souple et robuste, puisque chaque membre d'équipage peut se substituer à un collègue qui est en retard ou qui a des difficultés, ou encore déceler des erreurs potentielles de mesure ou de communication. Faire circuler les représentations entre plusieurs personnes permet ainsi de la valider naturellement. Les marins ne se comportent pas comme des automates exécutant des tâches fixes, mais comme des agents proactifs attachés à faire réussir l'ensemble du processus, et donc intéressés à la performance de leurs collègues :

> De plus, ils reconfigurent dynamiquement leurs activités en réponse à l'évolution des exigences de la tâche. Cela équivaut à une restructuration de systèmes fonctionnels qui transcende les membres individuels de l'équipage. Ceux-ci accomplissent leur tâche en construisant des systèmes fonctionnels locaux qui coordonnent les médias de leur environnement immédiat. Ils doivent également coordonner leurs activités de s'aider les uns les autres dans cette coordination. Le calcul est implémenté dans la coordination des états de représentation, et les participants humains coordonnent leurs actions de coordination les uns avec les autres[14].

C'est donc l'équipage tout entier, ainsi que les outils spécifiques à son activité de navigation, qui est la « machine ». Il n'y a pas d'instance centrale (le commandant par exemple) vers qui toute l'information remonterait afin qu'elle en tire une décision que les entités périphériques n'auraient plus qu'à exécuter, celles-ci jouant un simple rôle d'interface entre elle et le monde, comme dans le modèle du système physique de symboles présenté par Simon. Le système de règles est distribué et il prend la forme d'un réseau de contraintes communicationnelles auquel se soumet

13. (Hutchins 1995, p. 219).
14. (ibid., p. 219).

l'équipage pour résoudre les problèmes :

> Loin d'être une simple trajectoire à sens unique pour l'information, la communication est en fait l'assemblage de plusieurs types de contraintes bidirectionnelles, du bas vers le haut et du haut vers le bas. Les significations des assertions et des questions ne sont pas données dans les énoncés eux-mêmes mais sont négociées par les participants dans le cadre de leur compréhension des activités en cours. Les participants utilisent des suppositions sur les tâches des autres pour résoudre les ambiguïtés dans la communication. Des interprétations significatives particulières des énoncés sont simultanément proposées et présupposées par les cours d'actions qui les suivent. La preuve que chaque participant a d'une communication réussie est le flux de l'activité conjointe elle-même[15].

Les règles ont donc la forme de *contraintes*. D'une manière générale, une contrainte ne dit pas au processus ce qu'il doit faire, mais ce qu'il ne peut pas faire. Pour Hutchins, telle est la forme générale des procédures humaines : elles déterminent rarement un ordre précis d'exécution des tâches, mais établissent des dépendances séquentielles plus lâches. Elles disent par exemple, que telle tâche ne peut commencer que si celle-ci est terminée.

Il n'est parfois pas même besoin de le préciser, lorsque cette dépendance est inscrite dans les caractéristiques physiques de l'environnement, ainsi par exemple, que les chaussettes doivent être enfilées avant les chaussures, car l'inverse n'est pas possible. Les procédures humaines s'appuient souvent sur de telles caractéristiques externes pour se présenter naturellement à l'agent. Ainsi, la *check-list* d'inspection d'un avion avant le décollage énumère les items à vérifier dans l'ordre où ils vont apparaître à l'inspecteur qui fait le tour de l'avion, dans un certain sens. Ou encore, lorsqu'il s'agit de tâches cognitives coopératives, les activités de l'un peuvent être contraintes par celles d'un autre qui lui fournit des informations nécessaires. C'est en ce sens fondamental qu'il est pertinent de parler d'une écologie cognitive : l'environnement tout entier de chaque membre d'équipage – l'environnement naturel, les instruments du bord, sa propre position sur le navire, ses collègues ainsi que leurs sollicitations, la hiérarchie militaire, les autres symboles sociaux et culturels – tout cela contraint et *règle* naturellement son activité, déchargeant d'autant le besoin de règles explicites. Ces derniers se trouvent donc focalisées sur les points *non évidents* de la procédure, notamment ceux où plusieurs choix justifiés sont possibles, comme par exemple la fréquence de vérification par un officier supé-

15. (ibid., p. 236).

rieur de la position[16].

On peut à présent montrer la cohérence des deux assertions apparemment opposées de Hutchins. D'une part, le système tout entier, composé de l'équipe de navigation et de l'ensemble de l'écologie cognitive qui l'enveloppe peut tout à fait être dit calculer. Caractériser des procédures par un réseau de contraintes n'a rien d'extraordinaire dans un cadre informatique, puisque c'est ainsi, on s'en souvient, que procède la programmation logique*[17]. Les système de production* de Post, dans leur forme leur plus générale, fonctionnent exactement ainsi, et Hutchins a donc raison de les évoquer.

Cependant, ce système calcule tout en minimisant les calculs et les transformations complexes de représentation *pour chaque membre de l'équipage*, et plus généralement, le besoin de recourir à une procédure écrite pour accomplir sa tâche. Une procédure écrite est en effet un artefact coûteux pour les capacités cognitives humaines. Il s'agit d'un point sur lequel Hutchins est particulièrement convaincant, anticipant ici les résultats actuels des neurosciences[18]. Suivre une procédure écrite, en effet, requiert de coordonner un acte de lecture avec la mémorisation de la position de l'instruction courante, la représentation de celle-ci (à la fois la situation qu'elle décrit et les actions qu'elle prescrit) et enfin sa traduction en actions motrices. Elle oblige l'agent à coordonner, à chaque étape, au moins trois espaces, celui des séquences de symboles, celui des représentations mentales, et celui de l'action dans l'environnement immédiat. Cette coordination « spatiale » (d'un espace à l'autre) empêche naturellement la coordination « temporelle » (au sein de chaque espace), notamment la coordination motrice et la compréhension (la visualisation du flux dynamique des représentations). C'est ce que parvient à réaliser l'exécution experte, au contraire, qui peut enchaîner directement les actions et les représentations, car ces flux ne doivent plus être guidés, pas à pas, par la grille symbolique de la procédure.

Les procédures sont donc une partie minime mais essentielle de l'écologie cognitive de la navigation : elles servent à attirer l'attention des marins sur des points délicats des opérations à effectuer, ou sur des contraintes à respecter qui ne sont pas déjà matérialisées dans les instruments, ou dans les relations spatiales et culturelles entre marins. En particulier, elles servent à allouer les tâches entre les membres de

16. (Hutchins 1995, p. 180).
17. Voir plus haut, § 37.
18. (Hutchins 1995, p. 295-310). Sur les neurosciences, voir le développement suivant.

l'équipage lors de procédures exigeant une forte coordination : chacun sait par elles quel rôle il doit tenir, mais surtout quels rôles doivent tenir également ses collègues, qui agissent comme des contraintes pour lui. Un phénomène de coordination par convention se met ainsi en place[19].

C'est pour ces raisons que Hutchins peut affirmer que si les marins se contentaient de suivre les procédures écrites, rien ne fonctionnerait. Prises à elles seules, celles-ci décrivent un réseau largement *sous-contraint*, qui laisse ouvert mille possibilités de faire s'échouer le navire. Les comportements qu'imposent l'environnement matériel et surtout la situation sociale des marins (la hiérarchie militaire, le désir de reconnaissance par le groupe) en sont le soubassement implicite. Par exemple, Hutchins cite la coutume, nulle part écrite, de proposer son appui à son équipe dès qu'on a terminé sa tâche car « faire le strict nécessaire lorsque d'autres savent qu'on a du temps et des moyens pour en faire davantage est un message clair [qu'on leur envoie][20] ».

C'est tout cet ensemble de « règles » qui font du navire un système de production. Mais il est clair qu'il ne s'agit pas ici de règles auxquelles *obéissent* les marins, mais de *régularités* que relève l'ethnologue qui décrit leur comportement. Hutchins conclut donc son enquête fermement de la manière suivante :

> Mon hypothèse initiale sur le travail en milieu militaire était que les comportements y sont explicitement décrits et que les gens agissent plus ou moins comme des automates. Il devrait être évident maintenant que c'est loin d'être le cas[21].

Nous allons à présent présenter la conséquence principale de l'étude de Hutchins pour notre propos avant de revenir à la raison pour laquelle il maintient néanmoins, contre toute attente, la métaphore informatique.

§ 80. La notion d'écologie cognitive

L'analyse de Hutchins accomplit un pas essentiel. En mettant en évidence le rôle très limité des procédures écrites dans les opérations de navigation, elle montre que les règles explicites sont rarement le véhicule privilégié de l'action, que celle-ci n'aurait qu'à emprunter pour être amenée à destination. Il vaut mieux les comparer, dans le même registre, aux balises qu'on trouve sur les chemins de randonnée, souvent espacées de plusieurs centaines de mètres, qui permettent de contrôler

19. (ibid., p. 375).
20. (ibid., p. 225).
21. (ibid., p. 225).

régulièrement qu'on ne s'est pas égaré, qui marquent le chemin lorsqu'il devient incertain, mais qui *supposent* par ailleurs une certaine capacité du randonneur à le suivre sans leur appui.

Les procédures sont donc des *aides* à la réflexion pratique humaine plutôt que son substitut, ainsi qu'on le croit trop souvent. Comme les balises, qui sont insérées dans l'environnement afin d'empêcher le promeneur de se fourvoyer, elles invitent le praticien à ne pas « se laisser aller » à son mouvement naturel (soit qu'il soit encore un apprenti, soit qu'il y ait une difficulté ou un piège spécifique à la situation présente), et à *prendre le contrôle* de son action. Clark le montre assez bien lorsqu'il écrit :

> Le rôle du langage structuré comme outil pour échafauder l'action a été exploré dans diverses littératures [...] Les exemples tirés de l'expérience quotidienne sont abondants et vont d'instructions mémorisées au laçage de chaussures jusqu'aux formules répétées avant de traverser une rue, comme « regarde à gauche, à droite, à gauche encore, et si la voie est libre, traverse avec prudence ! » [...] Dans de tels cas, l'agent qui utilise le langage ainsi [...] peut s'engager dans une sorte simple d'auto-échafaudage comportemental, utilisant la séquence phonétique ou spatiale des codes symboliques comme guides de la séquence temporelle des actes[22].

La procédure pour traverser une rue ne consiste pas à *automatiser* l'action de l'agent, mais au contraire à l'inviter à réfléchir avant de prendre une décision – ici prêter attention à son environnement visuel. De la même manière, quand on décide de poser un calcul plutôt que le faire de tête, c'est qu'on veut prêter une attention particulière à nos opérations. Les règles écrites et les procédures sont des artefacts spécialement conçus pour guider l'attention de l'agent et l'obliger à *contrôler* consciemment son action, à désactiver l'action automatique et habituelle dans certaines circonstances précises.

La psychologie cognitive contemporaine interprète ces phénomènes par l'opposition entre le système 1, celui de l'intelligence habituelle et automatique, et le système 2, celui de l'attention et de la réflexion capable de « désactiver » le système 1 dans certaines circonstances[23]. Suivre une procédure symbolique pas à pas est clairement une opération pilotée par le système 2, potentiellement dans le cadre d'un entraînement du système 1 à acquérir de nouveaux automatismes. Les procédures sont particulièrement adaptées à cette fonction, non parce qu'elles sont faciles, mais justement au contraire parce qu'elles ont un coût cognitif élevé, qui

22. (Clark 2010, p. 48).
23. (J. Evans et Frankish 2009 ; Kahneman 2012).

mobilise entièrement l'esprit. Suivre une procédure est l'inverse d'une action automatique et aveugle, comme la métaphore de la machine nous induit faussement à le croire. C'est sur cette idée que Dehaene, un neuro-scientifique, conclut son étude de l'apprentissage des mathématiques :

> Je ne surprendrai guère le lecteur en avouant que je soupçonne la métaphore de l'ordinateur d'être insuffisante et excessivement limitative. Sur un plan purement empirique, la comparaison ne tient pas. Les chapitres précédents abondent en contre-exemples qui suggèrent que le cerveau humain ne fait pas de calculs à la manière d'une machine logique. Les calculs rigoureux ne viennent pas facilement à l'*Homo sapiens*. Comme tant d'autres animaux, il vient au monde avec une idée floue et approximative des nombres, qui n'a rien de commun avec la représentation digitale des ordinateurs. L'invention d'un langage des nombres et d'algorithmes de calcul exact appartient à l'histoire culturelle récente de l'humanité, et à bien des égards c'est une évolution contre nature. Tandis que notre culture inventait la logique et l'arithmétique, notre cerveau demeurait inchangé et rétif aux algorithmes les plus élémentaires. Je n'en veux pour preuve que la difficulté avec laquelle nos enfants assimilent les tables arithmétiques et les règles du calcul. […] L'échec de la métaphore du cerveau ordinateur est presque comique. Là où excelle l'ordinateur, l'exécution sans faille d'une série d'étapes logiques, notre cerveau se révèle lent et faillible [24].

On oppose souvent la pratique à la réflexion, en insistant que la seconde serait une entrave à la première, soit parce qu'elle ralentit l'action routinière, soit parce qu'elle empêche l'intuition informulée du geste juste, qui naît avec la véritable maîtrise. Cependant, entre ces deux pôles de la pratique (la routine et la maîtrise intuitive), l'attention et la réflexion accompagnent l'action selon un continuum de modalités. En particulier, il est parfois important pour l'action d'être guidée par le contrôle explicite de l'attention. Il s'agit là de l'un des rôles essentiel des procédures comme artefacts cognitifs.

§ 81. Les interprétations organisationnelles de l'informatique

Afin d'interpréter ce résultat, il est intéressant de comprendre la manière dont Hutchins le fit lui-même, en maintenant la métaphore informatique pour décrire le comportement collectif de l'équipage du navire.

Pour cela, il faut d'abord remarquer que Hutchins, en 1995, n'emprunte pas son idée d'une écologie cognitive au mouvement du même nom (*ecological cogni-*

24. (Dehaene [1995] 2010, p. 257-258).

tion), qui était seulement naissant au début des années 1990. Il ne cite pas James Gibson, le psychologue de la perception qui en fut l'inspirateur, mais bien Herbert Simon.

En effet, dans une métaphore célèbre, Simon remarque qu'une fourmi, sur une plage battue par le vent, dessine un trajet fort complexe, « une séquence de segments irréguliers et anguleux – pas vraiment aléatoire, car on y trouve une direction sous-jacente, dirigée vers un but », qui ne s'explique pas par la complexité de son raisonnement ou par celle des lois de son comportement, mais tout simplement par les obstacles qu'elle rencontre sur son chemin, auxquels elle s'adapte un à un.

> Vu comme une figure géométrique, le chemin de la fourmi est irrégulier, complexe, difficile à décrire. Mais cette complexité est à vrai-dire celle de la surface de la plage, pas celle de la fourmi[25].

Hutchins, en faisant sienne cette métaphore, souscrit à l'idée simonienne que la rationalité se situe à l'interface entre l'organisme et son environnement, et que celle-ci est donc *distribuée*. Bien que nous n'ayons pu revenir dans les analyses consacrées à Herbert Simon[26] sur le développement historique de sa réflexion, il faut se rappeler que son premier grand ouvrage, *Administrative Behavior*[27] montre comment un comportement rationnel *collectif* peut émerger d'une organisation humaine soumise à des structures et des règles élémentaires. Pour Hutchins, cette thèse reste valide même si on récuse la thèse subséquente de Simon, que l'organisme est lui-même une machine. Il a conscience également de l'enrichir significativement en mettant l'accent sur le rôle actif que joue l'environnement matériel dans ce comportement collectif (organisation de l'espace, instruments, environnement naturel) mais cela ne l'amène pas non plus à mettre en question l'hypothèse fondamentale sous-jacente qu'une pratique, qu'elle soit individuelle ou collective, *est rationnelle dans la mesure où on peut la décrire comme un calcul, c'est-à-dire comme une transformation de représentations*.

La pratique rationnelle est donc d'abord un phénomène collectif avant d'être un phénomène individuel. Tout en récusant l'idée que la personne humaine serait elle-même une machine, Hutchins garde celle que son comportement rationnel *émergerait* de son interaction avec son environnement matériel et social, ou encore

25. *SA*, p. 51.
26. Voir plus haut, sections 4.1 et 4.2.
27. (Simon [1947] 1997).

qu'il serait largement sculpté par le réseau de contraintes auquel il doit s'adapter[28].

Cette perspective, que Hutchins appelle *connaissance distribuée* (*distributed cognition*), s'inscrit dans une interprétation du rôle et de la valeur de l'informatique qui se développa dans le courant des années 1990, et qui conçoit les ordinateurs comme des machines à communiquer et à transformer des communications. En effet, si la rationalité réside dans l'interaction des personnes entre elles et avec leur environnement, alors l'effectivité des ordinateurs consiste à s'insérer dans cet écosystème afin de le faciliter ou d'augmenter son effectivité.

Hutchins, ainsi, s'investit dans des domaines de recherche appliquée en plein développement à cette époque, concernant l'*interaction homme-machine* (en anglais : *HCI, human-computer interface*) et le *travail coopératif assisté par ordinateur* (en anglais : *CSCW : Computer-Supported Cooperative Work*), qui ont l'ambition de contribuer à la conception de logiciels adaptés aux pratiques professionnelles grâce à des études d'ethnographie du travail. Pour donner un exemple issu de son étude des cockpits d'avion, Hutchins s'élève contre le remplacement du cadran de vitesse par un affichage numérique, en montrant que les pilotes se préoccupent peu de la vitesse exacte de l'appareil, mais qu'ils intègrent cognitivement la position approximative de l'aiguille à des manœuvres diverses. L'aiguille est un indicateur visuel bien plus efficace qu'une série de chiffres, et à partir de ce constat il propose un cadran numérique qui augmente cette efficacité, grâce aux possibilités nouvelles offertes par l'informatique[29].

Une esquisse de l'histoire du *CSCW* est réalisée par Schmidt (2011, chapitre 11.2). L'idée centrale que l'étude de Hutchins nous a permis de mettre en avant, à savoir que les procédures humaines ne sont pas des proto-programmes informatiques[30], est prise comme point de départ pour tenter d'adapter l'informatique à la pratique humaine. Le *CSCW* a des ambitions plus grandes que la conception d'interfaces homme-machine, puisqu'il s'agit de comprendre comment l'ordinateur peut aider les personnes au travail à mieux résoudre ensemble leurs problèmes. Il ne nous appartient pas ici de juger les contributions de cette recherche à la conception effective de logiciels. Schmidt (ibid., chapitre 11.4) en dresse un tableau plutôt mitigé ; aujourd'hui le *CSCW* est toujours actif, mais semble s'être réinventé comme discipline de l'informatique sociale (*social computing*) et se situe plutôt

28. (Hutchins 1995, p. 705). Voir également (Hollan, Hutchins et Kirsh 2000, p. 176).
29. (ibid., p. 184).
30. (Schmidt 2011, p. 111, 342).

dans l'orbite du *HCI*.

En tous les cas, il ne semble pas que cette fonction de facilitation permette d'expliquer à elle seule l'effectivité des ordinateurs dans notre monde. Il faut certainement adapter l'ordinateur aux pratiques humaines *si on décide que celles-ci doivent rester centrales*. Cependant l'effectivité des ordinateurs prend souvent la forme d'un décentrement, où la fonction des tâches accomplies par les personnes sont automatisées, et où celles-ci disparaissent du processus ou sont reléguées à sa périphérie. En particulier, dans les domaines administratifs d'aujourd'hui, les occupations humaines consistent à assister les ordinateurs à effectuer leurs tâches plutôt que l'inverse. Les ambitions humanistes de ces chercheurs sont importantes et louables, mais elles n'ont pas d'intérêt direct pour notre sujet : elles combattent ce que nous cherchons à comprendre.

Par ailleurs, la difficulté théorique de cette perspective sur l'informatique se voit dans une autre manière d'interpréter la métaphore de Hutchins, que la rationalité pratique pourrait être comparée à un système computationnel distribué. Ainsi Holt (1997), un théoricien de l'informatique, développe à la même époque une philosophie de l'information complète autour de la notion d'activité organisée, c'est-à-dire à la fois coordonnée et régulière – une notion qui ressemble à une esquisse des *pratiques* de MacIntyre[31]. L'information décrit des états de choses, en tant qu'elles participent au *plan* d'une activité organisée[32]. C'est pour cela que l'information (et les ordinateurs) sont essentiels aux pratiques humaines, parce qu'elle permet de mieux planifier nos activités, *i.e. décider*[33]. On pourrait citer, dans le même esprit de l'époque, le *Frisco Report*[34], un travail collectif entrepris à la fin des années 1990 pour clarifier l'espace conceptuel des *systèmes d'information*. Pour ces auteurs, toutes les pratiques peuvent être modélisées par des structures imbriquées de triades *acteurs-actions-actants* qui comprennent aussi bien les ordinateurs que les personnes humaines et les choses.

De telles interprétations sont cependant aux antipodes de l'esprit dans lequel Hutchins conçoit sa métaphore. En effet, pour Holt, l'activité organisée n'est finalement rien d'autre que sa performance selon un plan :

> Réaliser une activité organisée signifie : suivre son PLAN tout en tolérant les EXCEPTIONS. La théorie dit donc que toute activité organisée est décrite par un plan, mais

31. Voir plus haut, § 74.
32. (Holt et Cardone 1999, p. 84).
33. (ibid., p. 87).
34. (Falkenberg et al. 1998).

> que son exécution réelle peut impliquer (et implique généralement) des déviations par rapport au plan. De toute évidence, les pratiques de création, de gestion, de modification, etc. d'une activité organisée sont elles-mêmes une activité organisée[35].

En d'autres termes, Holt reste ici dans une conception cognitiviste classique, celle de Miller, Galanter et Pribram ([1960] 2013) déjà évoquée[36], pour qui l'activité intelligente est essentiellement l'exécution d'un plan ou d'une procédure. Tout le travail de Hutchins vise à montrer le contraire.

Ce hiatus vient de ce que la métaphore computationnelle, même appliquée à un groupe humain organisé plutôt qu'à ses membres individuels, se révèle malgré tout peu adaptée à l'idée centrale d'une écologie cognitive. D'abord, les règles qu'évoque Hutchins, comme celle d'aider son collègue dès que possible, ne sont pour la plupart pas effectives : elles sont floues et requièrent une interprétation, ce qui les disqualifient pour figurer dans un modèle computationnel. Par ailleurs, dire qu'un système physique calcule requiert que les observables qu'il transforme, ainsi que leur loi de transformation, puissent être décrits indépendamment de leur matérialité – comme des grandeurs numériques ou des schémas symboliques réplicables par d'autres systèmes de calcul[37]. L'idée d'écologie cognitive est diamétralement opposée à ce principe, puisque c'est par l'installation matérielle d'artefacts et par leur interprétation partagée que le système forme et fait évoluer ses représentations.

De facto, c'est bien de cette manière que le travail de Hutchins sera interprété par les cognitivistes de la nouvelle génération – ceux de la cognition « 4E » – et progressivement Hutchins abandonnera la métaphore computationnelle : dans son article de 2010, intitulé de manière significative *Cognitive Ecology*[38], il se réfère bien à James Gibson et à ces nouveaux courants cognitivistes plutôt qu'à Herbert Simon. Il ne parle presque plus de cognition distribuée.

5.2 La raison pratique contre la rationalité procédurale

À ce stade de la réflexion, nous arrivons à une position presque diamétralement opposée à celle de Simon. Loin que l'action humaine suive inconsciemment

35. (Holt et Cardone 1999, p. 82).
36. Voir plus haut, § 9.
37. Nous suivons ici Piccinini (2015), dont nous examinons les thèses plus loin, section 11.5.
38. (Hutchins 2010).

des procédures, ou même que le savoir dont elle dérive puisse être adéquatement décrit par elles, elle nous semble au contraire répugner à ce type d'artefacts et n'y recourir qu'en des circonstances très particulières, lorsqu'il s'agit d'assurer une attention maximale à la situation et à ses décisions – soit qu'il faille se méfier de ses propres automatismes, soit que la situation soit délicate et potentiellement piégée, ou lorsqu'il s'agit de se soumettre à un savoir étranger qui n'explique pas ses raisons.

Comment donc Simon a-t-il pu penser quelque chose de si éloigné de ces évidences ? Et surtout, que faire de notre question directrice, qui se trouve par ces conclusions renvoyée à son point de départ, voire davantage empêtrée ? En effet, si le rôle des règles relativement aux actions humaines est celui de balises qui guident de loin en loin un chemin, comment expliquer que, dans leur version informatique, elles se trouvent avoir une grande effectivité ?

On pourrait être tenté de répondre que cela est réalisé grâce à la bien plus grande précision dont sont dotées les règles informatiques, si bien qu'elles pallient les « trous » laissées par les règles humaines. Pour filer la métaphore du chemin, les règles informatiques seraient tout simplement des balises bien plus rapprochées les unes des autres, qui guideraient le marcheur « pas à pas », de telle sorte qu'il n'ait plus besoin du moindre sens de l'orientation. Comme nous l'avons noté en introduction, une telle interprétation des procédures comme de méthodes *plus précises* est courante[39].

Une telle explication n'est pas suffisante. D'abord, les programmes informatiques ne tirent pas leur effectivité de leur seule imitation des comportements humains, qu'ils se contenteraient d'exécuter plus vite et moins cher. S'il ne s'agissait que de cela, on n'observerait qu'un accroissement de productivité des activités humaines, et non pas les bouleversements évoqués en introduction.

Par ailleurs, nous avons vu au chapitre précédent qu'il était simpliste de réduire la programmation à la mise en place d'une imitation informatique d'opérations mentales. C'est un fait bien connu que les programmes informatiques arborent souvent un comportement qui, quoique correct logiquement, entre en conflit avec les attentes intuitives des usagers, ou même les désespère. L'interprétation contractuelle de la programmation montre qu'elle peut *imposer* à des usagers de modifier en profondeur leur comportement. Cela ne serait pas possible si elle ne faisait que connecter les règles plus lâches et plus libérales destinées aux personnes par des

39. Voir plus haut, § 18.

règles plus minutieuses destinées aux machines.

L'objectif de cette section est d'observer *directement* des situations de conflit entre les pratiques humaines *rationnelles* et des procédures rigoureuses (informatiques ou non) afin de comprendre comment *diffèrent* leurs formes de rationalité.

Un second exemple d'ethnologie du travail en milieu technique, concernant la maintenance de photocopieurs professionnels, nous permet de comprendre la limite des livres de procédures pour résoudre des problèmes. Lorsqu'ils échouent dans leur rôle – par exemple parce que la situation rencontrée n'y est pas prévue – ils s'avèrent peu efficaces pour aider la réflexion des techniciens à résoudre le problème par eux-mêmes. Pour transmettre leur expérience les uns aux autres, ceux-ci préfèrent aux procédures les *récits* d'investigations passées et les représentations *causales* (§ 82).

Autrement dit, les procédures ont une tolérance faible à l'ambiguïté inhérente des situations, contrairement aux personnes pour qui l'ambiguïté est souvent une ressource, une source de flexibilité accrue. Nous donnons quelques exemples de cette tension entre la rationalité pratique des opérateurs humains et les programmes informatiques (§ 83), et montrons qu'elle n'est pas due à la rigidité des outils informatiques (même si cela peut jouer un rôle) mais à l'exigence de systématicité et de cohérence qui est au fondement même de tout projet logiciel (§ 84).

§ 82. L'insuffisance des livres de procédures

Nous avons montré à la section précédente que les procédures ne guident que de manière limitée l'action humaine en milieu technique. Ici, nous voulons montrer qu'elles ne sont même pas la forme privilégiée par laquelle la réflexion pratique s'exprime, que ce soit pour progresser dans son investigation ou pour établir un dialogue avec d'autres personnes. Cette conclusion nous semble dériver clairement de notre deuxième étude concernant les opérations en milieu technique, celle de Orr ([1996] 2016), qui observa l'activité d'une équipe de maintenance de photocopieurs professionnels Xerox au début des années 1990.

Orr présente un résultat très similaire à celui de Hutchins concernant l'impossibilité d'expliquer le comportement des techniciens par la simple obéissance à des règles :

> On croit généralement que le travail technique consiste en la réparation routinière de machines similaires. La réparation de routine est en effet courante. Cependant, les

> machines individuelles ont parfois un comportement très particulier. De nouveaux modes de défaillance apparaissent régulièrement, or une procédure routinière ne peut pas résoudre des problèmes inconnus. La pratique des techniciens [se comprend mieux comme] une réponse à la fragilité des appréhensions disponibles des situations problématiques de service et à la fragilité du contrôle de leur définition et de leur résolution. […] Le travail dans de telles circonstances résiste à la rationalisation, car l'expertise indispensable à une pratique aussi contingente et improvisée ne peut être aisément codifiée[40].

De ce fait, leur rapport aux livres de procédure est ambivalent. Ils ne sont qu'un outil parmi d'autres dans l'exercice de leur travail, et ils déclarent s'en méfier :

> Le discours des techniciens sur la documentation de service abonde de mises en garde sur les dangers des procédures de diagnostic. Certaines d'entre eux sont dites circulaires ; d'autres sont facilement mal interprétées. Même les techniciens qui professent le plus fort attachement aux procédures de diagnostic avertissent que, sans comprendre l'intention de chaque procédure, on peut facilement faire le mauvais choix et se retrouver désespérément égaré[41].

Orr observe que la forme essentielle de partage des connaissances entre techniciens n'est pas la procédure mais le récit (*story*) : les techniciens, quand ils sont ensemble, racontent leurs aventures de service, les difficultés rencontrées, leur réaction et leur stratégie de résolutions, ce qui a fonctionné et n'a pas fonctionné, l'attitude du client, etc. jusqu'à la résolution finale. Pour Orr, le récit a certainement une valeur de socialisation (montrer sa compétence, échanger des émotions, partager des opinions communes sur tel client) mais on ne peut le réduire à cela. Il a également une valeur épistémique propre qui est reconnue par les techniciens et même par les responsables d'équipe, qui commencent toujours leurs réunions en retard pour laisser le temps à la discussion informelle[42]. Ainsi, ce sont ces récits qui sont mobilisés lorsqu'un technicien présente à ses collègues un problème qu'il ne sait pas résoudre.

La préférence du mode du récit sur celui de la procédure générale pour véhiculer des informations concernant la résolution de problème peut intriguer. Il ne s'agit pas seulement d'une préférence pour le concret sur l'abstrait, et de l'exemple sur l'idée générale, au contraire : ces techniciens, comme on va le voir, sont très friands d'explications générales et bien entendu capables de raisonnements abs-

40. (Orr [1996] 2016, p. 2).
41. (ibid., p. 129).
42. (ibid., p. 140).

traits. En particulier, s'ils sont certes bricoleurs, ils ne se contentent pas de solutions dont ils n'ont pas l'explication. Leurs valeurs communes gravitent autour des idées de contrôle, du bon ordre de marche des choses, et de systématicité dans le travail.

Orr insiste particulièrement sur ces derniers points :

> Deux problèmes principaux pour les techniciens sont la fragilité de la compréhension et la fragilité du contrôle. […] Le discours des techniciens révèle qu'ils valorisent au plus haut point les attributs qui contribuent au maintien de l'ordre et de la compréhension. […] La meilleure riposte à cette menace de chaos est, croit-on, d'avoir une approche systématique du travail[43].

Cette approche systématique ne réside pas seulement dans la méthode du diagnostic, qui ne doit écarter *a priori* aucune hypothèse. Il s'agit également de précision et de propreté des gestes, de rangement de ses affaires et des pièces détachées, du suivi d'un certain protocole de visite, de gestion de ses clients et de son territoire d'intervention.

On peut donc être surpris qu'un tel état d'esprit de groupe se contente de l'échange de récits et ne cherche jamais à en tirer des leçons systématiques. C'est la résolution de ce paradoxe qui nous intéresse dans l'enquête d'Orr.

La raison de la complexité du diagnostic est d'abord tout simplement le nombre de parties dont est composé un photocopieur professionnel, ainsi que le mélange de pièces mécaniques et électroniques. L'ensemble baignant dans une poussière ininterrompue d'encre, par moment chauffée à de très fortes températures, soumis à des cadences élevées, on conçoit que l'usure et l'encrassement peuvent être rapides et concerner n'importe quelle partie de la machine. Par ailleurs, la fréquence d'entretien, le suivi par les utilisateurs des consignes d'usage, les incidents déjà connus par la machine, la variabilité apportée par les réparations antérieures et par les pièces de rechange – tous ces facteurs expliquent que chaque situation peut être différente et que chaque photocopieur a son histoire propre. Orr rapporte le fait que les techniciens connaissent leurs machines *individuellement*, ainsi que leur « perversité » ou leur « docilité », comme un pasteur connaît ses brebis[44].

Chaque diagnostic d'une panne complexe est donc une enquête qui doit tenir compte de cet arrière-plan constitué par le contexte d'usage et par l'histoire de la machine. Cette enquête doit aboutir à la *compréhension* de la panne, c'est-à-dire

43. (ibid., p. 163).
44. (ibid., p. 89).

à la représentation du mécanisme défaillant et de ses causes :

> Les techniciens doivent se préparer à résoudre des problèmes nouveaux et non anticipés, ce qui requiert d'eux qu'ils développent une compréhension aussi complète que possible de la machine. […] La question de l'interprétation est critique ; les techniciens travaillant sur des diagnostics incomplets disent souvent qu'ils croient connaître les faits cruciaux mais ne les reconnaissent pas. La partie la plus difficile consiste à reconnaître l'importance d'un fait donné relativement au comportement observé ou rapporté de la machine[45].

On comprend, à partir de là, que les vertus épistémiques du récit sont particulièrement adaptées à la nature des problèmes rencontrés. Il permet d'abord d'entremêler continument deux histoires, celle de la machine et celle de l'enquête, en les mettant en regard l'un de l'autre. Il permet également de conserver les détails pertinents aussi bien que ceux qui ne le sont pas, mais qui ont pu retarder le diagnostic. Enfin, il met l'auditeur en situation et lui permet d'être lui aussi acteur de l'enquête qui se déroule dans le récit ; elle aiguise sa propre capacité de jugement. Ultimement, le récit devient un outil de diagnostic *en situation*. Il permet de comparer, avec tout le détail requis, la situation présente avec le récit. Mieux encore, il constitue un *modèle* qu'on peut tenter de projeter *sur* la situation présente :

> Les récits sont des artefacts à faire circuler et à préserver. À travers eux, l'expérience devient reproductible et réutilisable. En même temps, chaque récit est, en un sens, une re-représentation. Les récits prennent naissance dans des situations problématiques et sont racontés ou répétés pendant le diagnostic, lorsque l'activité qu'elles représentent devient problématique à nouveau.[…] De tels récits combinent des faits sur la machine avec le contexte de situations spécifiques. Les informations contextuelles démontrent la validité revendiquée des faits du récit et fournissent un cadre pour tester ces affirmations relativement à la machine présente. Le contexte contraint également l'application de ces assertions[46].

Les récits sont donc les moyens de la compréhension et de sa transmission. Mais la compréhension elle-même porte sur le mécanisme interne de la machine et sur le point précis de sa défaillance. Ce que visent les techniciens à travers tous leurs récits, c'est l'identification de ce mécanisme défaillant, la cause de son apparition, et l'explication de ses symptômes. En d'autres termes, le procès de l'investigation est un travail de représentation d'une réalité cachée *derrière* des phénomènes dont elle est la cause ou la raison, qu'on pourrait comparer à celle d'un observateur de

45. (Orr [1996] 2016, p. 109, 119).
46. (ibid., p. 127).

phénomènes de la nature. Il est néanmoins plus exact de rapprocher la posture des techniciens à celle des ingénieurs qui ont construit la machine :

> Cette capacité à décomposer la machine en ses éléments constitutifs, ou à composer la machine entière à partir d'eux, est caractéristique de la façon dont les techniciens parlent de leurs machines et peut-être pensent à elles[47].

Cette posture se voit d'ailleurs dans l'admiration générale que les techniciens éprouvent pour les machines bien conçues, et leur désir de les voir bien fonctionner. Ils sont d'ailleurs souvent enthousiastes à propos des nouvelles machines[48]. C'est cette posture qui est en partie à l'origine de leur approche systématique du travail. Leur enquête, idéalement, ne doit pas seulement résoudre le problème, mais également en donner la raison. Les pires problèmes sont en effet ceux qui sont à répétition, car ils montrent que la réparation précédente n'avait soigné que le symptôme et non la cause[49].

On peut à présent percevoir pourquoi récits et approche systématique ne sont pas antithétiques dans le contexte d'un diagnostic complexe. Le récit est la forme même du diagnostic qui *cherche* à être systématique, à *cerner* le problème :

> Pour les diagnostics difficiles, la représentation du problème prend la forme d'une narration du processus de diagnostic, d'une considération verbale de ce que l'on sait ou pense savoir ou de ce que l'on pourrait savoir de la situation, pour voir si elle peut être interprétée de manière cohérente. Le récit consiste à résumer ce que l'on sait sur la machine, à se demander si ce qui est connu est une représentation cohérente de la situation ou pourrait l'être s'il était interprété différemment, et à déterminer ce qu'il faudrait savoir de plus et comment y parvenir[50].

Tout cela permet d'expliquer la relation complexe et ambivalente qu'ont les techniciens vis-à-vis des manuels de maintenance. D'une part, même s'ils n'apprécient pas l'idée d'être réduits à un rôle de simple exécutant, ils apprécient l'approche systématique des procédures, qui correspond à leur éthique de travail. Mais ce qui les intéresse prioritairement dans les manuels sont les schémas de la machine, et les énumérations des pièces détachées. C'est cela qui leur permet de construire une image mentale du mécanisme et de ses points de dérèglement potentiels. Par contraste, les procédures proprement dites de réparation sont considérées avec méfiance, car elles supposent très souvent une situation de panne beaucoup trop

47. (ibid., p. 92).
48. (ibid., p. 25).
49. (ibid., p. 71).
50. (ibid., p. 109).

naïve, qui n'est d'ailleurs le plus souvent pas explicitée : elles se contentent de caractériser les problèmes par les symptômes (tel message d'erreur, etc.) et de mettre en regard une série d'actions à mener. La vie des techniciens avec leurs machines est bien plus complexe que cela.

Ce que les techniciens reprochent donc aux livres de procédures n'est pas d'être trop directifs, mais de n'être pas assez systématiques. D'ailleurs, il n'est pas vraiment question de reproche, car ils pensent qu'on ne peut prévoir à l'avance tous les cas de figure possibles[51]. La seule systématicité possible est celle de l'investigation, et celle-ci requiert d'avoir une claire compréhension de la machine pour pouvoir *raisonner* à son propos[52]. La fausse systématicité qu'offre la documentation est suivie à contre-cœur, lorsque le technicien se trouve démuni de toute autre solution : au moins le met-elle à l'abri de tout manquement professionnel[53].

Ces observations sont très révélatrices pour notre enquête concernant le rôle des procédures et des règles dans l'action humaine. Si on met de côté pour l'instant leur aspect dévalorisant – le sentiment d'être un simple exécutant – elles apparaissent surtout comme incapables de fixer le savoir nécessaire à la résolution de problème, par opposition d'une part aux récits, qui ont l'avantage de sauvegarder le contexte et le procès de l'investigation elle-même, et d'autre part à la compréhension du mécanisme interne de la machine.

Nous avons déjà développé le premier point : en quelque sorte, le récit donne la proto-histoire de la procédure, celle de la réflexion qui identifie la cause génératrice de la panne et la solution réparatrice. La procédure apparaît en ce sens comme la simple *trace* d'une investigation réussie, mais qui n'en transmet pas le savoir essentiel, celui qui est utile à la résolution de problèmes futurs, et qui doit plutôt porter sur la manière de trouver son chemin.

Nous revenons sur le second point – le besoin de comprendre le mécanisme interne de la machine – en fin de chapitre.

§ 83. Le rôle de l'ambiguïté dans l'action humaine

L'exemple canonique des *workplace studies* consacrées aux relations hommes - machines est donné par Lucy Suchman dans son étude de 1987, inspirée de l'ethno-

51. (Orr [1996] 2016, p. 111).
52. Orr utilise l'expression *reasoning* pp. 106 et 108. Il utilise plus souvent *to think about*, par exemple p. 24, 92, 105.
53. (Orr [1996] 2016, p. 110).

méthodologie de Garfinkel et portant (également !) sur les photocopieurs Xerox. Cette étude s'oppose de manière frontale à l'idée de la psychologie cognitive computationnelle (et à Herbert Simon en particulier) que l'action s'expliquerait par des plans et des compositions de règles. Nous ne revenons pas sur cette question déjà traitée à la section précédente, d'autant plus que l'argument de Suchman est discutable[54]. L'intérêt de l'étude de cas de Suchman pour notre propos actuel, relativement à celles de Hutchins et de Orr, est qu'elle porte directement sur l'interaction des personnes avec un livre de procédure « en action », c'est-à-dire un menu interactif censé guider l'usager du photocopieur dans la programmation de sa tâche comme, par exemple, faire cinq copies reliées d'un document relié, recto-verso, avec agrandissement.

Suchman montre, par l'enregistrement et l'analyse minutieuse des interactions entre la machine et les personnes (qui étaient associées par deux, afin de favoriser l'explicitation orale de leurs raisonnements et de leurs difficultés) que des quiproquos opérationnels peuvent très aisément se produire, car d'une part la machine n'a pas accès à l'état cognitif des personnes, et d'autre part celles-ci peuvent interpréter de travers les messages de la machine. Par exemple, si l'usager interprète de manière fautive une instruction de la machine de fermer le capot (alors qu'il s'agit de fermer la couverture du document, car *cover* a ce double sens en anglais), et que la machine lui demande par conséquent de le ré-ouvrir, l'usager va interpréter cela comme l'indication qu'il l'a mal fermé, ou trop brutalement.

Les exemples donnés par Suchman peuvent faire sourire. Ils datent de la préhistoire des réflexions sur les interfaces homme - machine, et les *designers* ont fait depuis des progrès extraordinaires sur ces questions. Cependant ils illustrent un fait qui reste fondamentalement valide : une procédure, même interactive, ne peut interpréter la situation que de manière prédéterminée en fonction des informations dont elle dispose, et n'a pas *le sens* de ce qui se passe. De nos jours encore, nous pouvons être exaspérés par le manque d'à-propos des serveurs vocaux interactifs. L'élément essentiel de Suchman est que les personnes, lors d'une interaction, disposent de ressources implicites pour calibrer entre elles la signification commune qu'elles accordent à la situation. Le programmeur, au contraire, réalise un *pari* sur les attentes des interlocuteurs futurs de la machine, et une telle négociation de la signification ne lui est pas disponible. Ainsi, il ne peut vérifier que l'utilisateur a bien compris le message, et de son côté, ce dernier n'a pas la possibilité de poser

54. Voir ici la discussion critique de Schmidt (2011), chapitre 12, p. 359.

des questions sur l'état actuel de la procédure (ce que Suchman appelle une *méta-investigation*), comme par exemple de savoir si le silence de la machine est une validation ou un blocage. Ainsi il peut y avoir de fausses alarmes (« la procédure a déraillé ») ou au contraire l'installation d'un quiproquo sur une longue séquence d'interactions[55].

L'intérêt de cette analyse – il faut insister là-dessus – est qu'elle ne porte pas sur les difficultés qu'on relève habituellement sur l'interaction homme - machine. Il ne s'agit pas par exemple de l'incapacité de la machine à percevoir les émotions de ses interlocuteurs humains. Les difficultés portent sur des décalages proprement logiques concernant l'interprétation à donner *aux procédures elles-mêmes*.

Il semble donc y avoir, dans les interactions homme-machine, ce que nous aimerions appeler une *ambiguïté* essentielle concernant la situation. Il ne s'agit pas seulement d'une ambiguïté sur les mots et les symboles, comme celle qui est apparue dans l'exemple plus haut, mais d'une ambiguïté à propos des objectifs à remplir, des moyens à disposition, de l'état de préparation de chacun des protagonistes. Aussi ne se manifeste-t-elle pas seulement, ni même essentiellement, dans le dialogue homme-machine et dans leurs interactions de surface, mais dans le disjointement de leurs logiques respectives d'action.

Ce disjointement est visible de manière privilégiée dans les *évolutions de besoin*, terme par lequel on désigne les demandes émises par le maître d'ouvrage de faire évoluer un logiciel. Ces demandes sont, comme nous l'avons vu[56], une partie importante de l'activité de l'équipe informatique au service d'une organisation. La plupart d'entre elles sont bénignes et peuvent être résolues en quelques jours ; mais certaines se révèlent complexes à satisfaire, parce qu'elles touchent à des points sensibles de l'architecture interne du logiciel. Cela ne veut pas dire qu'elles sont complexes à formuler, et elles peuvent apparaître souvent comme de bon sens.

Une évolution de besoin peut être due à de multiples facteurs. Il peut s'agir de corriger une spécification initialement fautive, si l'ambiguïté s'était infiltrée en son sein même, par exemple si le maître d'ouvrage avait passé sous silence un besoin qui lui semblait aller de soi, et dont parfois il n'avait pas même conscience. Il peut s'agir aussi d'une évolution de la situation elle-même. C'est à ces cas-ci que nous allons nous intéresser, en donnant quelques exemples tirés de notre expérience

55. (Suchman 1987, p. 120).
56. Voir plus haut, § 30.

personnelle.

1) La production de verres ophtalmiques se fait en partie à la commande, puisque chaque verre complexe est taillé en fonction de la vue du patient. Il existe des paramètres de production permettant d'optimiser la minceur du verre, mais qui peuvent présenter d'autres défauts. Aussi certains clients (les opticiens) ont-ils des préférences différentes sur ce sujet. Le système informatique permettait d'enregistrer cette préférence, si bien que ces paramètres étaient activés ou non en fonction du client à l'origine de la commande. Lorsque ce logiciel fut implanté en Allemagne, l'équipe commerciale de ce pays demanda à l'équipe informatique de permettre à l'opticien de spécifier *pour chaque commande* si les paramètres d'optimisation devaient être activés ou non. Les opticiens allemands, connus pour leur minutie, souhaitaient pouvoir déroger *dans des cas précis* à la règle générale qu'ils avaient fixé.
Une telle demande était légitime mais elle informatiquement complexe. Elle requérait que la préférence inscrite dans la fiche du client ne soit plus un paramètre fixe passé automatiquement à la commande, mais seulement une *valeur par défaut* qui devait pouvoir être modifiée lors de la prise de commande (par exemple une case à cocher ou décocher). Une telle évolution n'est pas évidente, car faire évoluer un modèle de données ainsi qu'un formulaire de commande sont des tâches qui ont des impacts sur l'ensemble de la chaîne de transmission des informations. On a donc là une demande du maître d'ouvrage parfaitement justifiée et très simple à comprendre, qui pose des problèmes importants à l'informatique. Il s'agit d'un cas d'ambiguïté typique : Il est parfaitement naturel pour une personne de vouloir *parfois* varier sa préférence, une notion qui, par définition, implique au contraire une certaine stabilité.
2) Des problèmes de qualité avaient rendu les équipes commerciales nerveuses. Le cas (rare) s'était produit que la production à nouveau d'une commande défectueuse soit elle-même refusée par le client. La succession de deux problèmes de qualité pour une même commande sont désastreux vis-à-vis d'un client, aussi l'équipe de production souhaitait pouvoir procéder à des examens de qualité systématiques pour ces productions à nouveau : en fin de fabrication, si le produit était fabriqué localement, ou à réception, si celui-ci était fabriqué à l'étranger. Cela impliquait que ces commandes soient identifiées explicitement par un signalement spécial sur la fiche de production.
Cette demande, parfaitement compréhensible, était également très difficile à satis-

faire, car elle supposait encore de faire évoluer le modèle de données d'un ordre de production, et d'aiguiller systématiquement les ordres dotés de ce signalement vers le contrôle qualité, tâche qui n'était pas prévue par le logiciel de production, puisque le contrôle était seulement statistique. Là encore, il y a une situation d'ambiguïté, puisque, dans le modèle de données initial, il n'était tout simplement pas *prévu* de décrire un ordre de production comme *à nouveau* ou pas.
3) L'agence commerciale anglaise avait fixé, dans sa politique d'annulation de commande, un délai d'une journée complète après la prise de commande. Il était en effet à peu près certain que la production n'ait pas été lancée d'ici là, puisque celle-ci avait en moyenne deux journées de production en attente. Lors de la mise en place du nouveau logiciel qui automatisait l'ordonnancement de la production, se posa le problème des commandes passées le vendredi soir. Celles-ci ne pouvaient être annulées que le lundi (jour ouvrable pour le service client) alors que la production travaillait le samedi. Les annulations pouvaient donc arriver trop tard en proportion non négligeable. L'équipe de production, lorsqu'elle ordonnançait elle-même la production, avait l'habitude de retarder celle des commandes du vendredi jusqu'au lundi suivant. Cependant l'ordonnancement automatique n'offrait pas la capacité de définir une telle règle. Il était par exemple capable de donner la priorité à certains clients importants, ou à certains types de produit, mais pas à certains *jours* de prise de commande sur d'autres.
Il y a, là encore, un exemple d'ambiguïté. Un paramètre non signifiant pour la décision peut devenir signifiant dans certains cas de figure, ici une différence d'organisation du travail concernant le week-end entre le service client et la production. Il est très simple pour une personne de suivre cette règle, mais pour un logiciel, il faut encore une fois mettre à jour son modèle de données pour qu'il puisse en tenir compte.

Ces trois exemples sont très similaires. Ils ne sont nullement représentatifs de *toutes* les formes d'évolution de besoin auxquelles un logiciel peut être confronté, cependant ils suffisent pour notre propos qui est de faire sentir le *décalage* essentiel qui existe entre un système informatique et une logique humaine de l'action. L'ambiguïté que nous avons relevée par trois fois ne se situe pas, comme dans la situation étudiée par Suchman, dans l'interface entre l'homme et la machine, mais dans les *règles* mêmes auxquelles il s'agit de se conformer.

Aussi est-il notable que les ambiguïtés décrites ne sont pas liées, comme semble le suggérer Suchman qui est inspirée par l'ethno-méthodologie, à l'aptitude spé-

cifique qu'ont les personnes à se comprendre entre elles à s'accorder sur le sens qu'il faut donner à la situation. Elles *peuvent* être résolues par une programmation complémentaire.

§ 84. Systématicité et adaptabilité

Ce point est important : il ne s'agit pas de dire qu'il existe un domaine de la rationalité humaine qui serait inaccessible aux règles et à la programmation – question sur laquelle un scepticisme de bon aloi est suffisant ici – mais de pointer vers la différence spécifique qu'il y a, pour l'homme et pour le programme, à s'adapter à l'ambiguïté.

Cette différence peut sembler mineure : il semble tout simplement que, dans les cas cités, la résolution de l'ambiguïté se trouvait être *plus difficile* pour le programme que pour les personnes, qu'il y avait un décalage entre la simplicité et le bon sens de la demande du maître d'ouvrage et la difficulté technique qu'éprouvait le programmeur à la satisfaire.

Cette difficulté n'est cependant pas anodine. Elle est due, dans les trois cas cités, au besoin de faire évoluer le *modèle de données* du programme, c'est-à-dire la représentation que se fait le programme de la situation. Il est bien connu des informaticiens que là se situe le point critique d'une bonne ou d'une mauvaise programmation, comme l'affirme cet adage de l'informaticien Fred Brooks : « La représentation est l'essence de la programmation[57] ». Ce qui est essentiel ici est la notion de *modèle*. Comme nous l'avons vu, toute résolution informatique d'un problème doit passer par une modélisation globale de la situation et de toutes ses dépendances pertinentes[58]. Chaque évolution de besoin peut potentiellement requérir de reprendre à zéro cette modélisation, car l'ajout d'un seul paramètre peut remettre en cause l'ensemble des hypothèses du modèle : on retrouve ici le problème du cadrage[59].

Ce mode d'intégration de nouvelles règles à un système existant est spécifique de la programmation et plus généralement de l'approche systématique d'un problème. Le terme d'*intégration* est lui-même significatif : il s'agit en effet de s'assurer que l'ajout d'une nouvelle règle préserve l'*intégrité* du nouvel ensemble. Il ne s'agit pas là d'une précaution superfétatoire, mais de *ce que programmer veut dire* : *as-*

57. Voir plus loin, § 201 pour la citation complète et la référence.
58. Voir plus haut, 3.3.
59. Voir plus haut, § 9.

surer la résolution automatique d'un problème général, à travers *tous* les cas de figure qui conviennent à sa spécification.

Ce n'est pas du tout ainsi qu'agissent les personnes. Si on instruit un opérateur d'une nouvelle règle à suivre, il va tout simplement essayer de s'y conformer, sans mettre à jour un quelconque « modèle mental ». Il va s'y adapter *localement*, si on peut dire, et non reconfigurer un système global de comportement. Dans le cas où un conflit de règles apparaît et qu'il s'en aperçoit, il posera une question à son responsable. Il est fréquent qu'il ne s'en aperçoive pas et qu'il fasse un choix implicite. Si celui-ci pose problème, le responsable devra établir un diagnostic, etc. L'action humaine est ainsi faite naturellement d'une gestion *a posteriori* des conflits d'ordres et de règles, que la programmation et l'automatisation viennent justement réformer, en proposant une gestion *a priori* – ce qui est bien audible dans le *pro-* de *programmer*.

La facilité relative qu'ont les personnes à comprendre et à intégrer de nouvelles règles, voire à gérer leur conflit de manière habile, ne vient donc nullement d'une supériorité de l'intelligence humaine sur celle de la machine, qui serait obtuse, rigide, balourde, comme on l'entend souvent. Le défaut de cette facilité est sa non-systématicité, c'est-à-dire le risque qu'elle prend de *mal* résoudre le problème. Son mode opératoire est celui de la gestion d'exceptions, c'est-à-dire de la possibilité de reconfigurer le problème au cours de sa résolution si la situation s'avère inédite.

Cette forme d'intelligence situationnelle ne doit pas être opposée à la balourdise de la machine, ce qui n'a aucun sens, mais à une autre modalité de l'intelligence humaine, qui est de prévoir. L'une n'est pas supérieure à l'autre. Il y a toujours un arbitrage délicat entre automatisation et gestion *ad hoc*. Laisser certaines règles au jugement des opérateurs peut être légitime et avantageux dans certains cas, notamment lorsqu'il s'agit de décisions qui doivent pouvoir être justifiées. Si on reprend le problème de la production du samedi dans la filiale anglaise, un ordonnancement manuel permet de gérer plus finement les priorités – par exemple si l'ordonnanceur sait, d'expérience, que tel client n'annule jamais de commandes, s'il y a des urgences, etc.

Néanmoins, les directeurs d'usine sont les premiers demandeurs d'automatisation des décisions le long du flux de production. Ils savent que dans un système de production complexe composé de centaines de règles, on ne peut en confier que quelques-unes à l'appréciation de chaque poste de travail. Pour le comprendre, il suffit de considérer que plusieurs opérateurs peuvent occuper un même poste, qu'il

faut tous les former, qu'il y a régulièrement de nouvelles recrues, etc.

Il ne faut donc pas confondre rigueur et rigidité. La rigueur imposée par la programmation la rend plus rigide à court terme et localement, c'est-à-dire du point de vue de l'opérateur qui voit bien qu'il pourrait prendre une décision plus intelligente *dans ce cas précis*. À grande échelle cependant, dans le cadre d'un système complexe, le taux d'exceptions à gérer peut devenir rapidement inacceptable et asphyxier le flux de production lui-même. C'est alors que l'automatisation libère, car elle permet de soutenir l'ajout indéfini de règles particulières tout en maintenant le taux d'exceptions sous contrôle. Dans ces cas de figure, la rigueur planificatrice est la condition de la flexibilité, et non son opposée.

Il est important de souligner que cette remarque de bon sens ne présuppose pas l'informatisation de la production. L'invention des outils de production flexibles dans le cadre de la révolution industrielle japonaise des années 1970-1980 n'a pas eu besoin d'informatique. Cela ne veut pas dire non plus qu'elle aurait été permise par une diminution des règles et des procédures relativement au modèle taylorien. Elle a contraire proposé une manière plus systémique de gérer la production, c'est-à-dire en prenant mieux en compte l'interdépendance des règles, qu'on a pu rapprocher de la cybernétique[60].

La rigueur qui impose une programmation systématique des règles n'est donc pas spécifique à l'informatique. Elle semble inhérente à l'idée qu'on peut se faire de tout outil industriel, et c'est cette proximité entre programmation et industrie qu'il faut à présent explorer.

5.3 Informatique et industrie

Dans cette section, nous rappelons d'abord (§ 85) la proximité historique et conceptuelle qui existe entre les champs de l'industrie et de la programmation, manifeste dans les réflexions de Babbage et de Lovelace dès la première moitié du XIX^e^ siècle. Cela nous amène à nous interroger sur la spécificité de la réflexion industrielle relativement à la simple réflexion technique (§ 86). Elle réside essentiellement dans la systématicité de l'analyse – division du travail, normalisation des tâches, des situations et des compétences – qui permettent de recomposer objectivement un processus de travail selon sa logique propre, et donc de *décentrer* l'opérateur humain vers la périphérie d'un dispositif conceptuellement maîtrisé qui est

60. (Dominici et Palumbo 2013).

la machine. La programmation, telle que nous l'avons appréhendée au chapitre 3, se situe donc dans le prolongement direct de ce mode de réflexion (§ 87).

§ 85. Proximité historique et conceptuelle

La proximité entre informatique et industrie se voit de manière frappante dans la proximité de leur champ lexical. *Processus, procédure, tâche, étape, instruction, exécution, machine, système, opération, spécification, règle, ordonnancement, planification, entrées, sorties, etc.* sont des termes centraux dans chacune de ces deux pratiques. Il s'agit certes là de termes assez généraux, mais on ne trouve pas un tel recouvrement lexical dans d'autres disciplines, par exemple le droit, la médecine ou même l'ingénierie. On pourrait même arguer que l'informatique a, globalement, a un champ lexical plus proche du monde de la production que de celui des mathématiques ou de la logique, auxquelles elle est souvent associée.

Cependant les termes que nous avons cités ont des dénotations matérielles très différentes dans chacun des deux domaines d'activité. Un processus informatique n'est pas un processus industriel, et une instruction informatique n'est pas une instruction que donne un contremaître à un opérateur. S'agit-il d'un transfert métaphorique d'un domaine à l'autre, lié au fait qu'on programme la machine justement pour *qu'elle imite* les tâches humaines ? Mais c'est ce rapport d'imitation qui est problématique ; on ne peut donc se contenter d'une telle réponse. Par ailleurs une métaphore fonctionne d'abord par la transposition d'un terme concret à un autre domaine spécifiquement distinct, or la plupart des termes mentionnés ici sont abstraits. Il se pourrait donc qu'on ait affaire à deux spécifications diverses d'un même champ lexical générique.

Qu'il soit métaphorique ou non, le transfert proprement dit de concepts industriels à l'informatique est attesté historiquement. Il ne se limite d'ailleurs pas à des transferts sémantiques. Le premier outil de représentation des programmes, les diagrammes de flux, fut proposé par John von Neumann en 1946. Or cette notation était devenue courante pour décrire les processus industriels dans les années 1920 – on l'attribue à Franck Gilbreth, un ingénieur de production situé dans la mouvance du *scientific management* de Frederick Taylor[61]. Il se peut que von Neumann les ait vus à l'œuvre par sa formation en chimie[62]. En tous les cas, pour lhistorien de l'informatique Mark Priestley, von Neumann et Goldstine, dans leur

61. (Kiran 2017).
62. (Ensmenger 2016, p. 326).

rapport de 1947 sur la programmation, conçoivent celle-ci sur le modèle de leur propre expérience concernant l'organisation du travail pour de larges salles de calculateurs humains :

> Ils reconnurent qu'il n'y avait rien dans leur description de l'étape de planification [du calcul] qui fût spécifique au calcul automatique, et les détails de leurs propositions reflétaient les pratiques établies dans l'organisation de l'informatique [humaine] à grande échelle [...] Cette continuité ne doit pas surprendre. Tous les développeurs des machines automatiques du début des années 1940 avaient une longue expérience dans la réalisation ou la supervision de calculs manuels à grande échelle. Ce qui était nouveau, c'était la nécessité d'être complètement explicite sur les opérations que la machine effectuerait[63].

Cette planification du calcul à grande échelle, fondée sur le principe de la division du travail et du contrôle de la qualité est une approche industrielle qui se perfectionne tout au long du XIX^e^ siècle[64]. Son modèle est le célèbre projet des tables trigonométriques mené à bien par Gaspard de Prony entre 1791 et 1795 à l'Observatoire de Paris. Des tables d'une précision inégalée sont produites en un temps record pour l'époque grâce à la décomposition des tâches de calcul en éléments simples à la portée de personnes n'ayant que de très communes compétences en arithmétique (des perruquiers au chômage suite à la Révolution française, selon la légende). Or Prony lui-même affirme avoir pensé à cette organisation en lisant le chapitre de *la Richesse des Nations*, d'Adam Smith, consacré à la division du travail[65].

La relation entre informatique et industrie est encore plus directe que cela, puisque Charles Babbage déclare avoir eu l'idée de sa *Difference Engine*, considérée comme la première conception de l'ordinateur, en lisant Prony qui avait écrit :

> Il me vint bientôt à la pensée d'appliquer à la connexion de ces Tables la division du travail, dont les Arts du Commerce tirent un parti si avantageux pour réunir à la perfection de main-d'œuvre l'économie de la dépense et du temps[66].

Babbage affirme ainsi :

> La division du travail peut s'appliquer avec autant de succès aux opérations mentales qu'aux opérations mécaniques. [...] Un bref exposé de son application pratique [...] montrera que les dispositions qui doivent régler l'économie intérieure

63. (Priestley 2017, p. 460).
64. (Grier 2005, p. 46 et suivantes).
65. (Durand-Richard 2010 ; Grattan-Guinness 1990 ; Peaucelle 2012).
66. Cité par (Babbage [1832] 2010, p. 154).

> d'une manufacture sont fondées sur des principes plus profonds qu'on ne l'a supposé, et sont susceptibles d'être employées utilement à préparer la route à certaines des investigations les plus sublimes de l'esprit humain[67].

Ce qu'il y a d'intrigant ici, c'est la conséquence que Babbage tire de sa découverte de l'applicabilité de la division du travail aux tâches de la pensée. On aurait pu s'attendre à ce qu'il conclue à l'importance de ce principe. Il en tire l'implication inverse, qu'il doit reposer lui-même « sur des principes plus profonds qu'on ne l'a supposé ». Lesquels ils sont reste mystérieux. Babbage se contente de pointer, de manière un peu grandiloquente, vers « les investigations les plus sublimes de l'esprit humain ». Cela laisse entendre que la division du travail repose sur un principe logique et non économique.

§ 86. La notion d'industrie

Afin d'interpréter ce lien entre informatique et industrie, ainsi que la suggestion de Babbage, il est nécessaire de clarifier en quel sens nous nous intéressons ici à l'industrie. Ce que ce terme recouvre est notoirement difficile à cerner. Une mise au point historiographique récente[68] passe en revue les principales caractéristiques discutées par les historiens et les économistes. La production à grande échelle, l'existence de normes permettant de reconnaître ou de juger les produits, l'écoulement de la production sur des marchés globaux, sont des caractéristiques assez larges pour qu'on puisse les reconnaître déjà dans des activités productives antiques et médiévales[69]. On trouve également dès ces époques une dynamique d'innovation visant le perfectionnement des produits et l'augmentation de la productivité. À partir du XIX^e^ siècle, l'industrie se définit par opposition à l'artisanat, en mobilisant tous les critères ci-dessus, ainsi que des notions complémentaires, de mécanisation (opposée au travail manuel) et d'énergie mécanique (opposée à l'énergie animale et humaine), de division du travail (opposée à l'unité des tâches de l'artisan), de prolétarisation de l'ouvrier (opposée à l'indépendance de l'artisan)[70].

On voit par la multiplicité et la nature de ces critères qu'il est difficile de circonscrire des classes précises d'activités économiques sous le terme *industrie*. Ainsi, il

67. (Babbage [1832] 2010, p. 154).
68. Revue *Artefact* (n°17, 2022).
69. (Pagès et Verna 2022).
70. (Caliste et Carnino 2022, p. 220).

serait réducteur de l'identifier à ce que les statisticiens appellent le secteur secondaire de la production de biens manufacturés. De larges pans de l'agriculture et des services sont aujourd'hui organisés selon une *logique industrielle*, ainsi les vastes cultures céréalières, les autoroutes ou les centres d'appels.

Qu'entend-on donc par ces expressions de *logique industrielle*, ou encore *méthodes industrielles*, qui sont devenues aujourd'hui courantes et qui intéressent particulièrement notre propos ? Les propositions des économistes contemporains[71] pour les circonscrire tournent autour de trois critères :

1. Une logique de standardisation des produits et des processus, qui permet l'organisation rationnelle du travail, l'automatisation, les effets d'échelle.
2. Une dynamique d'innovation.
3. L'importance du capital pour financer l'outil de production et l'innovation, ce qui suppose que l'activité est toujours à grande échelle.

Cette caractérisation n'est pas très éloignée de celle, apparemment beaucoup plus générique des historiens de la technique Caliste et Carnino qui proposent de définir le phénomène industriel « comme la croissance quantitative d'un phénomène technique au point où l'on assiste à une rupture qualitative[72] ». Ce qu'on doit entendre par *qualitatif* est très large et peut désigner la nature des procédés, l'organisation du travail, les marchés accessibles, etc. Cela permet néanmoins de répondre à l'objection qu'on pourrait faire à la définition de Le Blanc, que l'innovation a toujours existé dans les communautés artisanales. Pour Caliste et Carnino, l'industrie se caractérise par une rupture dans le mode même d'innovation, qui devient systématique dans les départements de R&D des entreprises[73]. L'intérêt de cette définition est de mettre le *phénomène technique* au centre de la logique industrielle, qu'elle permet de *reconfigurer* grâce à un changement d'échelle.

Le fait que l'industrie soit animée d'une dynamique d'innovation, et que cette dynamique elle-même soit organisée, met donc en valeur la dimension *réflexive* qui caractérise l'approche industrielle d'une activité technique quelconque. Cette approche est *technologique* dans le sens que ce terme a au XVIII^e^ siècle, d'étude des arts et métiers. Il est essentiel à l'industrie de sans cesse porter un regard *critique* sur ses produits, ses marchés, et ses procédés afin de les améliorer. Il n'y a ainsi pas de méthode ou de principe industriels qui aient une valeur absolue. Ure et

71. Voir par exemple (Le Blanc 2005, p. 29).
72. (Caliste et Carnino 2022, p. 232).
73. (ibid., p. 232).

Marx, par exemple, notent déjà que le principe de division du travail est mis à mal par l'automatisation qui *intègre* au contraire les tâches dans un seul flux de production.

Cela exige de comprendre la standardisation (le premier critère proposé par Le Blanc) comme autre chose que la simple homogénéisation des produits ou des activités. La variété, voire la personnalisation des produits, n'est absolument pas incompatible avec une approche industrielle, comme le montre l'opposition classique dans les années 1930 entre Ford et General Motors, cette dernière s'étant imposée grâce à la diversification de sa gamme d'automobiles. L'acte essentiel de la standardisation est la *spécification* qui permet, comme nous l'avons vu au chapitre 3, de déterminer *ce dont on parle* et de se mettre d'accord, *par anticipation*, sur le problème qu'il s'agit de résoudre. Donner une marque à un produit, c'est véhiculer à travers elle des promesses et des anticipations qui facilitent la transaction ; ce n'est rien d'autre qu'*imager* la spécification.

La division du travail doit être comprise selon le même principe. Ce que Ure et Marx critiquent, c'est l'organisation du travail tâche par tâche, qui n'est qu'une possibilité d'application d'une étude *analytique* du travail qui le résout en ses composantes élémentaires. Or une telle étude est tout autant nécessaire si on veut le mécaniser. Ce qui change fondamentalement avec les mercantilistes anglais, précurseurs d'Adam Smith sur l'idée de division de travail, est que l'augmentation de la productivité avec la spécialisation n'est plus seulement une donnée anthropologique – un fait déjà remarqué par Platon – mais « un levier ou un opérateur, […] un instrument économique à la disposition des entrepreneurs[74] ». C'est dans l'effort réflexif visant son optimisation que se situe l'essence de la division du travail. Elle est donc d'abord une opération *intellectuelle* que réalise l'ingénieur industriel observant le processus de travail avant de le reconfigurer. Cette étude revient à faire de ce processus un *objet* dont on peut mettre au jour les conditions – outils, gestes, force, précision, vitesse, etc. – pour éventuellement l'en libérer. C'est ce point de vue particulier qui marque la naissance de la technologie, dans le sens que lui donne Beckmann, ce professeur de sciences camérales de la fin du XVIII^e siècle lorsqu'il mobilise le terme pour désigner sa méthode de descriptions des métiers, non plus selon leur domaine d'activité, mais selon les fonctions élémentaires qu'ils réalisent :

> Le rabotage, technique du menuisier ; le polissage, technique du polisseur de glaces ;

74. (Séris 1994a, p. 15).

> battre le livre, technique du relieur ; presser les toiles ; calandrer le linge ; toutes ces opérations, aussi différentes qu'elles paraissent et aussi différentes qu'elles soient en réalité, n'ont qu'un but, à savoir le lissage des corps […] Or, je souhaite dresser un catalogue de tous les buts poursuivis par les artisans et les artistes au cours de leurs diverses opérations, ainsi qu'un catalogue de tous les moyens par lesquels ils savent atteindre chacun de ces objectifs. Je dénommerais un pareil catalogue la technologie générale[75].

Cette idée d'une taxonomie fonctionnelle des gestes techniques ouvre la voie à la considération abstraite des processus de travail et donc à leur recomposition générale, non seulement en fonction de la division du travail, mais également en fonction de leur objet. Là se trouve la révolution apportée par les machines selon Marx, de permettre la reconfiguration du processus de travail selon sa logique propre. Cela est au fondement de « l'extension du volume des machines motrices, de mécanisme de transmission et des machines motrices, [de la] plus grande complexité et diversité et [de la] régularité plus rigoureuse de leurs composantes à mesure que la machine-outil [prend] une configuration libre, uniquement déterminée par sa tâche mécanique[76]. » C'est bien cette leçon de Marx que développera Simondon à travers son concept d'*individuation*, et que Séris expose ainsi :

> Du point de vue […] de la rationalité technique […], celui qui voit opérer une machine industrielle [s'enthousiame] au spectacle de l'efficacité sans résidu d'une opération entièrement adéquate à sa finalité, mieux : entièrement conforme à son essence, à sa forme. La machine la débarrasse de tout accessoire inutile, elle la dépouille de tout ce qui n'est pas elle. S'adaptant d'emblée à l'échelle voulue, elle se fait géante pour opérer à l'aide d'organes hypertrophiés (la grue, la pelle mécanique) ou au contraire lilliputienne quand elle assemble et noue avec une infinie délicatesse et une rapidité saisissante les fils de soie les plus ténus.
>
> C'est la matérialisation de l'action à accomplir, réduite à sa forme pure, à sa simple silhouette ou à son profil intelligible, à sa rigueur, à l'exigence de son exécution rigoureuse. Elle uniformise, régularise, rectifie les gestes et les outils, plus encore qu'elle ne les réduit et les amplifie.
>
> Affranchie des contraintes « anthropomorphiques » du labeur humain, la machine est un travailleur collectif, le modèle auquel l'usine s'efforce de ressembler, quand elle se donne pour idéal d'être une usine sans travailleur. Le point de vue qui triomphe avec elle est celui de l'organisateur. […] Le dessin de la machine va

75. (Beckmann [1806] 2017, p. 67). Voir l'introduction de Carnino et Hilaire-Pérez à ce même ouvrage pour l'histoire des commencements de la technologie.
76. (Marx [1867] 2006, p. 429).

> pouvoir, bien loin de copier l'ouvrier et ses gestes, faire voir le jour à l'opération toute nue[77].

Nous avons cité ce texte en longueur car il permet de bien cerner la caractéristique centrale du concept d'*industrie* qui nous intéresse ici. L'industrie est liée aux machines non parce que celles-ci seraient une manière, historiquement datée, de substituer à des facteurs de production humains des facteurs physiques plus productifs, mais parce que les machines rendent possible l'expression libre du mouvement proprement réflexif qui caractérise l'activité industrielle, manifeste dans l'*innovation*, qui consiste à poser le processus de travail comme objet (*standardisation*) et à l'analyser (*division du travail*) en termes abstraits dotés d'une grammaire (*technologie*) afin de réagencer ceux-ci de manière optimale (*automatisation*).

Par cette caractérisation, nous ne prétendons nullement avoir cerné le phénomène historique de l'industrialisation, qui comprend également des aspects sociaux et politiques, mais seulement avoir mis en lumière sa dimension essentiellement transformative, que Carnino met au cœur de sa définition. Les activités industrielles désignent en effet tout type d'activités humaines reconfigurées dans le cadre d'un changement d'échelle, qu'elles soient productives ou non, lucratives ou non. Le commerce et la finance, l'administration publique et la santé, mais également les actions caritatives de grande ampleur sont toutes animées de réflexions industrielles, que l'informatique vient d'ailleurs éclairer et développer – nous y revenons au développement suivant. Il nous est important de préciser ce qui semble sous-entendu dans la définition de Carnino, à savoir que la reconfiguration industrielle d'activités à l'occasion d'un changement d'échelle est le résultat d'*actes de réflexion* à leur propos opérés par certains agents - qu'il s'agisse des entrepreneurs, des salariés organisés en coopérative, de l'État interventionniste, c'est-à-dire qu'elle procède d'une intention *rationnelle* d'amélioration incrémentale ou radicale. Cela est évident si on pense aux *crises* que la croissance non contrôlée d'une activité suscite naturellement : pénurie de main d'œuvre, de matières, de savoir-faire, engorgement de la distribution, etc. L'industrialisation n'est pas la cause mécanique de la croissance, mais la réponse rationnelle aux défis que pose cette dernière.

On peut résumer toutes ces explications du terme *industrie* par les trois oppositions suivantes :

1. L'industrie n'est pas liée au domaine de la *production*, mais aux *opérations*

77. (Séris 1994b, p. 159).

en général, telles que nous les avons définies[78].

2. L'industrie n'est pas liée aux machines dans le sens courant qu'on prête à ce terme. Des opérations impliquant seulement des personnes humaines peuvent être l'objet d'une approche industrielle. Cependant, il est caractéristique d'une telle approche qu'on demande alors aux personnes de se comporter *comme des machines*, c'est-à-dire d'obéir à des procédures précises. Ce point fait ressortir le lien profond qui existe entre les idées de *mécanisme* et de *procédure*, sur lesquelles nous revenons plus loin dans ce travail.
3. Une approche industrielle n'est pas seulement technique, elle est technologique. Les métiers explicitent et codifient leurs manières de faire depuis toujours. Ce qu'il y a de nouveau avec l'approche industrielle, c'est que l'étude technique, dans le contexte d'un changement d'échelle qui rend cruciales les questions de contrôle de l'action, devient systématique dans sa recherche de standardisation *et* d'innovation, c'est-à-dire à la fois dans la réduction analytique de la manière de faire à des éléments connus *et* dans sa reconfiguration de manière optimale.

§ 87. Une logique industrielle générale

On comprend dans ce cadre que l'informatique soit une activité industrielle par excellence. La programmation, telle que nous l'avons analysée au chapitre 3, n'est en effet rien d'autre qu'une démarche réflexive visant à résoudre un problème dans une situation donnée grâce au réagencement de ses termes opérationnels et à l'ajout de nouveaux termes opérés par la machine. Sa différence avec la définition qui vient d'être donnée de la réflexion industrielle tient dans sa généralité supérieure. Un problème, pour cette dernière, se limite à l'optimisation d'un processus de production, ce qui l'amène à concevoir le rapport entre la machine et les autres ressources de la situation (principalement l'opérateur de la machine) essentiellement comme l'imbrication de leurs opérations respectives. Les problèmes qui sont posés à la programmation, ainsi que les solutions qu'elle peut proposer, ne sont au contraire nullement contraints à ces formes restrictives – les exemples examinés jusqu'ici, empruntés à l'administration, aux jeux, au travail de la conception, le montrent assez. La programmation apparaît, à ce titre, comme la continuation et l'élargissement de la démarche industrielle, en tant qu'elle avant tout une posture d'analyse et de reconstruction d'une pratique humaine.

78. Voir plus haut, § 78.

De ce point de vue, ce ne sont pas des coïncidences historiques ni des transferts métaphoriques qui expliquent la proximité de l'informatique et de l'industrie, mais bien l'identité de leur posture épistémique qui subsume les aspects économiques (la division du travail) et techniques (l'analyse technologique, le machinisme) auxquels on associe le plus souvent la révolution industrielle. C'est cette posture épistémique unique qu'aurait aperçue Babbage, celle d'une « science des opérations » dont Ada Lovelace, sa célèbre collaboratrice, exprima parfaitement la possibilité :

> Il est sans doute désirable d'expliquer que, par le mot *opération*, nous entendons n'importe quel processus qui modifie la relation mutuelle de deux ou plusieurs choses, quelle que soit la nature de cette relation. Il s'agit là de la définition la plus générale qui soit, et qui inclurait tous les sujets dans l'univers. Dans les mathématiques abstraites, les opérations modifient bien sûr les relations particulières impliquées par les considérations du nombre et de l'espace [...] Mais la *science des opérations*, dérivée des mathématiques plus spécialement, est une science à part entière, et elle a sa propre vérité et valeur abstraites ; tout comme la logique a sa propre vérité et valeur, indépendamment des sujets auxquels nous appliquons ses raisonnements et processus. Ceux qui sont accoutumés à quelques-unes des vues modernes sur ce sujet sauront que, étant donné quelques relations fondamentales, certaines autres combinaisons de ces relations doivent suivre nécessairement, ces combinaisons pouvant être illimitées en variété et en extension si les déductions à partir des premières relations sont conduites assez loin. [...]
>
> Le mécanisme opératoire [...] pourrait agir sur d'autres choses que des nombres, si on trouvait des objets dont les relations mutuelles fondamentales pussent être exprimées par celles de la science abstraite des opérations, et qui fussent susceptibles d'adaptations à l'action de la notation et du mécanisme de la Machine. [...]
>
> La Machine Analytique est une concrétisation de la science des opérations, construite avec une référence spéciale au nombre abstrait comme sujet de ces opérations[79].

Il est possible à présent de dégager une perspective globale sur les manières dont l'informatique se rend effective dans les activités industrielles. Les exemples mentionnés au développement § 83 montraient l'inadéquation des logiques humaine et informatique de l'action. On voit à présent qu'elle n'est pas différente de celle, depuis longtemps relevée, entre les logiques humaine et industrielle. Toute machine, qu'elle soit informatique ou non, crée des situations d'ambiguïté avec les

79. Ada Lovelace, *Sketch of the Analytical Engine*, note A au mémoire de L.F. Menabrea (1843). Reproduit dans (Babbage 1989b).

opérateurs humains, car il n'est jamais clair si elle « saura » s'adapter à tel cas que l'opérateur juge « imprévu », ou encore, pour le dire autrement, si elle a la même notion d'imprévu que lui. Par ailleurs, la machine ne peut être adaptée au changement que par le détour d'une démarche réflexive complexe, peu courante chez les personnes. Cependant ces *défauts* sont compensés, du point de vue de l'effectivité, par la systématicité visée dans l'emploi de machines.

Mais que veut dire ici *systématicité* ? Ce terme a commencé d'apparaître dans nos analyses, sans que nous ayons vraiment cerné son sens. Or son emploi très générique dans les discussions portant sur nos sujets le désigne comme un problème complémentaire plutôt que comme une ressource pour notre enquête. L'interrogation sur ce terme va nous accompagner, d'une certaine manière, jusqu'au terme de ce travail. Ce qu'on peut faire à ce stade, dans le cadre d'une discussion des opérations en milieu industriel, c'est caractériser plus positivement les *avantages d'effectivité* liés à l'emploi de machines informatiques dans le cadre d'activités industrielles, en espérant que ces résultats contribuent à l'élucidation de ce terme. Ceci est l'objet de la section suivante.

5.4 Les formes d'effectivité de l'informatique

L'effectivité est une notion qui peut être envisagée à la fois de manière absolue et relative. Prise absolument, une solution est effective si elle résout un problème. Prise relativement, une solution peut être plus effective qu'une autre, relativement à certains termes du problème passibles de mesure, comme la proximité à l'objectif, le taux de succès, ou la consommation de ressources. Il ne s'agit pas nécessairement d'efficacité et de productivité, comme nous l'avons déjà noté[80]. C'est cette seconde forme d'effectivité que nous voulons étudier ici à travers le prisme de l'industrie, où elle se mesure donc comme *avantage*.

La notion d'avantage est délicate à manier, car elle est relative (entre autres) à un degré de contingence (avantage circonstanciel ou structurel), à une norme (la force, le profit, le bien-être, etc.), à un horizon de temps (à court-terme, à moyen-terme), à une certaine focale (le poste de travail, l'atelier, l'entreprise, le marché). Nous choisissons ici une focale médiane, qui permet de suivre concrètement les effets de l'insertion de machines informatiques dans une situation présentant une

80. Voir plus haut, § 3.

structure régulière stable. Nous parlons, selon la pratique courante des économistes, d'avantages durables contribuant à l'effectivité accrue d'un processus de travail relativement à des processus concurrents ne bénéficiant pas de cette insertion technologique.

Il ne s'agit donc pas ici d'explorer les changements structurels à grande échelle, économiques, sociaux et politiques qu'engendre le numérique. Nous avons mentionné en introduction les bouleversements des chaînes de valeur[81]. Du point de vue social et politique, le numérique modifie les rapports de force entre travailleurs et capitalistes. On peut se reporter ici par exemple aux analyses marxiennes de Casilli ([2019] 2021). Ce n'est pas à ces focales larges que nous nous attachons ici. Comme point de référence parmi les grandes analyses classiques des effets de la mécanisation, notre utiliserons plutôt la perspective adoptée par Charles Babbage dans son *Traité des Machines et Manufactures*[82] que celle, beaucoup plus ample, de Marx dans le chapitre 15 du *Capital*[83] consacré aux machines.

Notre thèse est que les avantages d'effectivité que procure l'informatique à des activités industrielles sont de trois formes : l'automatisation, la contrainte, et la traçabilité. Nous proposons d'introduire le sens qu'il faut donner à ces termes par un exemple simple, puisqu'il concerne une machine pré-informatique, la caisse enregistreuse qui se diffuse dans les commerces américains à partir des années 1870-1880 (§ 88). Nous accordons une attention spécifique à la notion de *traçabilité* (§ 89) avant de passer à un exemple beaucoup complexe, aujourd'hui encore en pleine transformation numérique, la gestion de chaînes logistiques (*supply chain management*) (§ 90).

§ 88. Les caisses enregistreuses

Les caisses enregistreuses furent inventées aux États-Unis dans les années 1870 par James Ritty. L'entreprise qu'il fonda, renommée *National Cash Register* (aujourd'hui NCR, toujours en activité) fut rachetée par John Patterson en 1884 qui en fit l'une des entreprises les plus innovantes, les plus agressives et les plus emblématiques du capitalisme américain de la fin du XIX^e^ siècle[84]. Pour les modèles les

81. Voir plus haut, § 6.
82. (Babbage [1832] 2010).
83. (Marx [1867] 2006).
84. Voir (Crandall 1988) pour une histoire de cette technologie, ainsi que (Friedman 1998) pour l'entreprise NCR.

plus simples, les caisses enregistreuses sont composées d'un clavier permettant de renseigner les montants des transactions, d'un rouage permettant de les additionner (voire de les imprimer sur un rouleau de papier interne), et d'un tiroir-caisse s'ouvrant uniquement après l'enregistrement d'une transaction. Ritty appela sa machine *the Incorruptible Cashier*, nom qui traduit assez clairement sa fonction initiale : prévenir le vol à la caisse par les employés qui y étaient préposés, puisque le commerçant pouvait chaque soir comparer les sommes présentes dans le tiroir-caisse avec les transactions enregistrées.

Cependant, ces outils se développèrent par la suite bien au-delà de cette fonction initiale, ainsi que le résume très bien cette monographie à leur sujet :

> Les premières caisses enregistreuses furent fortement promues comme un moyen de dissuasion contre le vol. Mais elles furent également les instruments d'une nouvelle approche analytique des affaires qui aboutit à l'utilisation systématique de l'information pour générer des profits. […]
>
> En 1902, vous pouviez obtenir une piste d'audit des transactions, un comptage des clients, des statistiques sur les montants détaillés et des totaux cumulés. Vous pouviez obtenir tout cela globalement mais aussi pour chaque vendeur. Vous pouviez même identifier les transactions de produits individuels. Cela permit de concevoir des promotions spéciales, des programmes d'incitation individualisés pour les employés et la collecte de données pour des études de marché quantitatives. Et, comme si cela ne suffisait pas, la caisse enregistreuse s'étendit au-delà de sa fonction éponyme (la gestion de l'argent liquide) vers la gestion du crédit. […]
>
> La caisse enregistreuse était la combinaison d'un système de sécurité, d'une machine comptable et d'un système d'information de gestion. Tout cela fut mis en œuvre avec des pièces mécaniques mobiles, des engrenages, des cames et des leviers logés dans de grands boîtiers. […] Ainsi, la caisse enregistreuse fut la force mécanique qui la première permit l'exploitation de l'information transactionnelle[85].

Les caisses enregistreuses jouent pour ces raisons un rôle important dans la protohistoire de l'informatique, avec les machines à calculer et les machines à tabuler[86]. *De facto*, les trois fonctions essentielles que leur reconnaît Crandall sont selon nous assez emblématiques de tout système informatique :

1. Compter et calculer, qui permet au caissier de gagner en productivité (« la machine comptable »).

85. (Crandall 1988, p. 3).
86. (Campbell-Kelly et al. [1996] 2014, p. 21).

2. Contraindre l'action du caissier, en l'empêchant par exemple d'ouvrir le tiroir de caisse hors du contexte d'une transaction (« le système de sécurité »).
3. Garder une trace des transactions effectuées avec un détail plus ou moins grand, pour différents usages, de contrôle ou de décision (« le système d'information »).

Ces trois éléments sont selon nous caractéristiques des différentes classes d'avantages que peut apporter l'informatique à une activité professionnelle. Nous allons les examiner tour à tour.

La première fonction est l'automatisation des tâches humaines, ici le calcul. Son bénéfice est double : nous avons déjà mentionné le gain de productivité du caissier, mais il y a également l'*exactitude garantie*. Il s'agit là un point très important : le gain de productivité n'est pas seule finalité possible de l'automatisation. Les réflexions de Babbage permettent d'éclairer ce point d'une manière générale.

La première section de son *Traité des Machines et Manufactures*, consacrée à établir une liste assez empirique des avantages procurés par la mécanisation, commence par mentionner la démultiplication de la puissance du travail humain, et l'économie de temps, autrement dit le gain de productivité en énergie et en temps, respectivement[87]. Cependant, il note déjà que certains effets mécaniques sont incommensurables avec le travail humain, comme l'usage d'explosifs pour dégager des rochers : la technologie nous permet de réaliser des tâches qui n'auraient tout simplement pu être envisagées manuellement. Il reprend ce thème à plusieurs reprises, notamment dans le chapitre 7 intitulé *Exercer des forces trop grandes pour la puissance humaine, et exécuter des opérations trop délicates pour la main humaine*[88]. Il y explique qu'une trop grande puissance requise ne peut être satisfaite par une armée de travailleurs ajoutant leurs forces. Leur coordination devient problématique au moment de l'effort, la gestion de l'espace également si la force doit être concentrée en un point précis, il y a enfin les difficultés annexes de la gestion d'un grand nombre de travailleurs (les recruter, les nourrir, les payer). Le titre du chapitre mentionne le cas inverse, celui d'opérations requérant une précision ou une cohérence du geste hors de portée d'un artisan.

Ce thème est repris au chapitre 9, qui y ajoute la similarité des produits, ce qu'on appelle aujourd'hui la stabilité ou encore la faible variance du processus industriel. Mais Babbage va encore plus loin, visiblement fasciné par une propriété

87. (Babbage [1832] 2010, p. 7-9).
88. (ibid., p. 35).

générale qu'ont les machines, de pouvoir garantir l'*identité des copies* d'une série d'objets manufacturés, et y consacre deux chapitres parmi les plus longs de son ouvrage :

> Rien n'est plus remarquable, et pourtant moins inattendu, que l'identité parfaite d'objets fabriqués par le même outil. Si le dessus d'une boîte circulaire doit être ajusté sur sa partie inférieure, cela peut être fait sur le tour en avançant progressivement l'outil du support coulissant ; le degré approprié de serrement entre la boîte et son couvercle est trouvé par essai. Après cet ajustement, si des milliers de boîtes sont fabriquées, aucun soin supplémentaire n'est nécessaire ; l'outil est toujours amené jusqu'à la butée, et chaque boîte sera adaptée de la même manière à chaque couvercle[89].
>
> […] Les deux dernières sources d'excellence dans le travail produit par les machines [précision et similarité] dépendent d'un principe qui s'applique à une très grande partie de toutes les fabrications et dont le bon marché des articles produits semble dépendre en grande partie. Le principe auquel il est fait allusion est celui de la copie, pris dans son sens le plus large. Des soins presque illimités sont, dans certains cas, donnés à l'original, à partir duquel une série de copies doit être produite ; et plus le nombre de ces copies est grand, plus le fabricant peut se permettre de prodiguer des soins et des soins à l'original[90].

Nous appellerons *exactitude* dans la suite ce travail cette propriété qu'ont les machines d'exécuter précisément et identiquement à travers le temps l'opération demandée.

Autrement dit, l'automatisation 1) permet des gains de productivité, 2) donne accès à de *nouvelles* opérations, qui ne sont pas envisageables sans elle, et 3) garantit l'exactitude du processus. L'automatisation des calculs par la caisse enregistreuse illustre donc le premier et le troisième type de bénéfices.

Elle n'illustre certes pas le second type de bénéfice, à part peut être l'activité de crédit citée plus haut. Cela est dû aussi bien au caractère très encadré des transactions commerciales de détail, qu'aux limites intrinsèques de sa construction mécanique. D'une manière plus générale, il est évident que la démultiplication incommensurable de nos capacités de calcul par l'informatique a profondément modifié, *qualitativement*, la plupart des disciplines qui y font appel – ainsi par exemple la science computationnelle[91].

89. (ibid., p. 48).
90. (ibid., p. 51).
91. (Humphreys [2004] 2007).

La seconde fonction de la mécanisation est la contrainte exercée par la machine sur une action tierce (humaine ou non). Dans le cas de la caisse enregistreuse, il s'agit de bloquer l'accès par le caissier à l'argent liquide. Babbage n'évoque ce type de fonction que par un seul exemple, celui des alarmes qui réveillent l'attention de l'opérateur, mais il n'est pas évident qu'il ait à l'esprit la régulation du travail en milieu industriel[92]. Les horloges à programme régulant les horaires des usines, des chemins de fer, etc., déjà mentionnées en introduction[93], datent de la fin du XIX^e siècle. L'informatique a permis de développer énormément cette fonction de contrainte pour des bénéfices de sécurité (droits d'accès à des ressources) ou de conformité à des règles. Nous avons déjà illustré ce point[94] par l'exemple de l'ERP qui force l'acheteur à choisir un fournisseur parmi ceux qui sont référencés, car le système ne procédera pas à la mise en paiement dans le cas contraire. Dans le cas de systèmes cyber-physiques[*], (comme l'est la caisse enregistreuse) la contrainte peut s'effectuer par une coercition physique, mais cela n'est pas toujours le cas. Dans le cas de l'ERP, elle s'exerce en fin de processus, quand l'acheteur va demander au comptable la mise en paiement de la facture. Il peut s'agir également d'alarmes, comme le remarque Babbage, ou de messages d'alerte, d'incitation, de menace, etc. Enfin, il est à noter que les règles auxquelles il s'agit de contraindre un usager ne sont pas nécessairement explicites ou même visibles pour lui. Un système informatique peut présenter des options différentes à différents usagers sans que ceux-ci soient au courant de cette limitation et de ses raisons.

Cette seconde fonction est centrale pour la réflexion développée dans ce chapitre. Les machines peuvent obliger la personne qui dépend d'elle (d'une manière ou d'une autre) à agir *comme une machine*. Dans une usine par exemple, l'écologie cognitive d'un opérateur est entièrement façonnée par les machines et leur logique propre, si bien que c'est le rythme et les besoins de la machine qui lui dictent ce qu'il doit faire. Les contraintes habituelles qu'on trouve dans une écologie cognitive (objets naturels, instruments et outils, signes et énoncés, environnement humain) doivent laisser une place importante, directe ou indirecte, à celles dictées par le processus mécanique. C'est là que peut émerger l'idée générale d'un comportement *mécanique* de la personne, un adjectif qui sera abondamment repris par les logiciens pour décrire la notion de procédure effective[95].

92. (Babbage [1832] 2010, p. 43).
93. Voir plus haut, § 13.
94. Voir plus haut, § 51.
95. Voir plus loin, § 193.

La troisième fonction de la mécanisation est la traçabilité de l'évolution de la machine, de ses interactions avec la situation, et surtout de la situation elle-même à travers ces biais. Dans le cas de la caisse enregistreuse, comme son nom même l'indique, il s'agit de garder trace des transactions effectuées dans un registre. Babbage a la prescience de l'importance de cette fonction, puisqu'il lui consacre tout le chapitre 8 de son *Traité*, intitulé *De l'enregistrement des opérations*. Il est aisé d'enregistrer par un dispositif mécanique les cycles de la machine, ce qui permet de mesurer la production ; ou encore, il est possible d'enregistrer certaines opérations humaines à des fins de contrôle (Babbage donne l'exemple de robinets de cuves de spiritueux), ou enfin de mesurer des quantités physiques, comme la distribution de gaz, la quantité de liquide restant dans une cuve, la distance parcourue, etc. On voit par là, qu'enregistrer des opérations peut *signifier* différentes choses, l'opération de la machine elle-même, celle de l'opérateur, ou l'évolution de la situation objective[96].

La proposition de nommer *traçabilité* cette fonction générale des machines est nouvelle. Cependant elle se justifie par l'importance grandissante que ce terme prend dans de nombreuses questions ayant trait au numérique. Il signifie d'abord la capacité à suivre des choses à distance par leurs traces numériques, par exemple le scan d'un code barre dans un entrepôt qui rend un colis « visible » au sein d'un réseau logistique. Cela permet notamment de garantir l'origine d'un bien et que son transport se conforme à certaines normes (par exemple la chaîne du froid pour un médicament). Par extension, *traçabilité* signifie aussi la capacité à suivre l'activité numérique des personnes, notamment sur Internet. Nous souhaitons donner ici un sens beaucoup plus large et générique à ce terme, et nous nous y attardons donc un peu, en lui consacrant un développement spécifique.

§ 89. La traçabilité

Crandall a bien raison de parler de la caisse enregistreuse comme d'un système d'information de gestion (*management information system*) puisque cette dénomination recouvre les outils informatiques (comme les ERP*) utilisés par les organisations afin de traiter les données liées à leur activité. L'information dont on parle ici est *opérationnelle* ou *décisionnelle*.

Le premier terme recouvre l'information qui décrit les éléments de la situation nécessaires à la conduite de l'activité, que ces éléments soient relativement

96. (Babbage [1832] 2010, p. 39).

stables (par exemple l'outil de production, les fournisseurs, les produits, etc.), ou qu'ils soient transactionnels (commandes, factures, transports, paiements, etc.). La fonction de la machine est de modifier des données qui lui sont fournies selon des règles et de les mettre à disposition de la bonne personne (ou agent robotique) au bon moment. Il peut s'agir d'une simple communication entre deux personnes (un email par exemple), du suivi d'un colis par une personne (comme on vient de le voir), ou de la coordination de machines comme une flottille de drones qui doivent rester coordonnés les uns avec les autres. Le système informatique médiatise tous ces échanges en temps réel en devenant comme une empreinte de la situation, cependant hautement mobile, capable de se réécrire au gré de ses évolutions. *Trace* doit donc être ici entendu au sens de Peirce, c'est-à-dire d'un signe pointant vers son signifié en vertu de la relation causale qui les lie. Une *donnée* renseignée, communiquée, archivée, traitée à travers les réseaux informatiques doit donc être ainsi comprise, très simplement, comme une trace qui se répercute d'événement en événement par des liaisons causales – ce qu'on appel un *signal*, à quelque niveau d'abstraction qu'on veuille le considérer.

Un signal n'est donc rien d'autre qu'une série de traces reliées les unes aux autres grâce à la causalité d'un médium (le circuit imprimé, la fibre optique, au niveau le plus concret), rien de plus qu'une trace d'événements successifs, qui sont sa seule signification pure, c'est-à-dire dénuée de toute interprétation. L'information, dans cette perspective, si on accepte sa définition courante par Berry, comme de données munies d'une interprétation[97], requiert d'abord de pouvoir correctement « remonter la trace » d'une donnée, en explicitant ses événements générateurs, afin qu'on puisse être assuré de son interprétation. Cela n'est pas toujours possible, puisque chaque transmission ou traitement d'une donnée est un événement qui vient créer une nouvelle trace à partir d'une ancienne, et qui vient donc s'ajouter à cette signification pure.

L'information, une fois exploitée opérationnellement, peut conserver une valeur *décisionnelle* : elle permet, comme le faisait déjà la caisse enregistreuse, de réaliser des statistiques régulières (quotidiennes, mensuelles, etc.) utiles au suivi de l'activité en général, mais surtout à la prise de décision, par exemple concevoir une promotion s'il s'agit de ventes en baisse, établir un plan de maintenance s'il s'agit de production défaillante, etc. Ici, ce n'est pas la propagation de la trace en temps réel qui importe, mais sa *conservation* dans le temps, et son appartenance

97. Voir plus haut, § 5.

à une série. *Trace* peut alors être entendue dans son sens mémoriel cumulatif, qui permet de reconstituer le parcours des éléments dynamiques de la situation, de la même manière que les traces d'un animal dans la neige dessinent son chemin. Des exemples sont les logs* d'une application informatique, les données de vente (cf. la caisse enregistreuse), les données de production (souvent construites à partir des logs* des machines industrielles), la navigation d'un usager d'Internet, le parcours GPX* d'un randonneur, etc.

La traçabilité désigne donc deux choses. Il s'agit d'abord du fait que la situation se trouve dotée d'une empreinte mobile qui se propage au sein de la machine par des séries de chaînes causales orchestrées – la trace est ici signal. Il s'agit ensuite de la capacité, pour un observateur, de disposer en un temps ultérieur d'une remémoration des dynamiques de la machine, de la situation et de leurs interactions. La traçabilité désigne donc, d'une manière générale, la fonction par laquelle une machine informatique « éclaire » une situation, c'est-à-dire la rend tout à coup disponible pour la communication, le traitement, et l'archivage, ou tout simplement pour la réflexion. Il importe de souligner qu'aucune connotation positive ne doit être ici accordée au terme *éclairage* : nous ne traitons ici que de l'intention quant à cette fonction, et sa réalisation peut être partielle, biaisée, fautive, etc. Le problème de la trace est qu'elle ne fait que marquer un événement causal passé, qui ne peut être qu'inféré et qui garde toujours ouverte la possibilité de sa falsification.

§ 90. La gestion des chaînes logistiques

Afin de tester cette hypothèse d'une tripartition des bénéfices fonctionnels de l'informatisation dans l'industrie, nous examinons dans ce développement une activité dont la transformation numérique est déjà profonde, mais loin d'être achevée, celle de la gestion des chaînes logistiques (*supply chain management*)[98].

Ce terme désigne, d'une manière générale, la mise à disposition des marchandises auprès de leurs destinataires (clients finaux ou distributeurs), qu'il s'agisse d'acheminer des produits finis jusqu'à eux, d'assurer la production de ces produits, ou encore de s'approvisionner en matières premières et en équipements. Toute entreprise manufacturière dispose d'une chaîne logistique, mais celle-ci peut être partiellement sous-traitée, notamment à des transporteurs, à des gestionnaires d'entrepôts, à des sites industriels et parfois à des prestataires opérant en totalité

98. Ce développement s'inspire de diverses sources, dont (Calatayud, Mangan et Christopher 2019 ; MacCarthy et al. 2016 ; Wu et al. 2016) donnent des points d'entrée.

des sous-ensembles de la chaîne logistique, par exemple dans des pays lointains, ou pour des produits spécialisés. Une grande entreprise peut aisément avoir plus de trois mille sites logistiques (production, entrepôts) et des centaines de milliers de commandes en transit par jour.

Une chaîne logistique est donc avant tout un système d'information qui doit assurer l'échange des informations pertinentes entre tous ces acteurs afin de garantir la *traçabilité* de ses flux. Cette traçabilité est d'abord opérationnelle, lorsqu'il s'agit d'établir des prévisions de livraison pour les clients, d'ordonnancer la production, d'identifier et de traiter les perturbations au sein de la chaîne (retards, blocages, pénuries, etc.). Pour ce faire, chaque colis est identifié par un code-barre ou une étiquette électronique permettant de le localiser et de le rendre visible au système d'information à chaque étape de transit. Par ailleurs les camions et les conteneurs qui les transportent sont eux-mêmes de plus en plus connectés, les rendant ainsi localisables et visibles en temps réel. Il est évident que seule l'informatique a rendu possible la constitution de chaînes d'une taille et d'une vélocité similaires à celles que nous observons aujourd'hui.

Pour certains produits sensibles, comme les denrées périssables ou les médicaments, la traçabilité des flux doit être effectuée à un niveau très fin et archivée, de manière à permettre leur inspection future. C'est cette propriété, dotée d'une portée juridique, qui donne son sens courant à ce concept.

La traçabilité est également décisionnelle. Certaines données opérationnelles sont archivées afin de permettre aux logisticiens d'établir des prévisions de besoins à plusieurs mois, de réserver les ressources nécessaires (transports, entrepôts, production), d'identifier des nouveaux flux ou des nouveaux produits à mettre en place, ou encore d'améliorer la performance de la chaîne logistique.

Le système d'information a un second rôle crucial, de *contraindre* le comportement des agents de la chaîne logistique, de telle sorte que soient livrés les bons produits aux bons clients, dans les délais et les coûts impartis. C'est au sein du système d'information que sont définis les flux de tâches* assignant à chaque agent les opérations à accomplir à un instant donné. Il peut s'agir par exemple d'ordres de production ou de livraison, d'instructions d'entreposage de colis reçus, de documentation administrative pour le transport international, etc. Le système d'information surveille la réalisation de ces tâches et déclenche des alertes si des retards sont constatés.

Enfin, le système d'information *automatise* de nombreuses activités logistiques,

à commencer par la gestion documentaire et les flux d'information eux-mêmes, qui autrefois passaient par des coordinations de personne à personne. Les tâches décisionnelles régulières de planification sont de plus en plus souvent automatisées. De nombreuses entreprises ont par exemple abandonné le processus traditionnel de prévision, qui dépendait trop des objectifs commerciaux, parfois irréalistes, de l'entreprise. Les logisticiens préfèrent aujourd'hui travailler avec des algorithmes statistiques exploitant les données historiques. Concernant les décisions opérationnelles, l'aiguillage des commandes vers les ressources adéquates (entrepôts, moyens de transport, sites de production) a tendance à être également automatisé, car cela permet de tenir compte des disponibilités de ces dernières en temps réel. Le trajet de la commande restant à parcourir est ainsi recalculé et optimisé à chaque point de décision, comme le fait un navigateur GPS pour les trajets individuels en voiture. Enfin, l'automatisation se fait *au sein même* des entrepôts, la robotisation de la manutention des marchandises permettant de nouvelles optimisations[99].

Ces conduites automatiques d'opérations ont un gain double de productivité. Elles permettent d'une part d'économiser du temps décisionnel et opérationnel humain, mais également et surtout de raccourcir les délais ou de réduire les coûts de la chaîne logistique dans son ensemble par des trajets, des ordonnancements et des stratégies de stockage optimisés. Elles rendent également possibles de nouvelles opérations, comme la fabrication à la commande déjà évoquée[100], ou l'audit complet de l'origine d'un bien.

Notre classification ternaire des avantages de l'informatisation dans l'industrie se trouve donc renforcée par l'examen de la gestion des chaînes logistiques. Il faudrait certes analyser plusieurs autres exemples pour nous assurer plus avant de sa généralité. Cependant, un argument déductif peut également être avancé. On peut en effet « lire » ces trois formes d'avantages d'effectivité dans le schéma fondamental de la programmation mis au jour au chapitre 3, consistant dans la double spécification du comportement de la machine et des agents de la situation. Elle correspond, respectivement, aux avantages de la *conduite automatique d'opérations* et de la *contrainte* exercée par la machine sur les agents. Quant à la *traçabilité*, elle se réfère d'abord à la capacité qu'a la machine d'interagir avec la situation, mais ensuite et surtout, à la dimension réflexive essentielle à la programmation, qui fait du programmeur ou concepteur également un *observateur* du processus

99. Nous illustrons plus en détail cette robotisation en § 229.
100. Voir plus haut, § 83.

qu'il a mis en place.

5.5 Le problème de la systématicité

L'idée générale dont nous sommes partis au chapitre précédent est que, si les techniques expriment le savoir-faire positif et discursif des pratiques humaines, et que leur explicitation prend la forme privilégiée de règles et de procédures, alors peut-être l'effectivité des ordinateurs pouvait s'expliquer par ce que la programmation en était le mode d'expression technique par excellence, puisqu'elle semble n'être rien d'autre que l'élaboration de systèmes de règles. Herbert Simon posa cette hypothèse aux fondements de son projet de *Sciences de l'artificiel*, par lequel il pensait pouvoir hausser les savoirs techniques au même degré de rigueur et de précision que les sciences naturelles, loin des « recettes » et des « traditions » de l'artisan.

Les conclusions des études de ce chapitre sont sans appel : la modalité par laquelle les réflexions industrielle et informatique résolvent les problèmes qui leur sont confiés diffèrent de la réflexion technique habituelle dans la même mesure que celle-ci diffère du savoir pratique immédiat de l'expert. Si la technique élabore des règles par une réflexion de second ordre qui a pour objet la réflexion pratique immédiate, alors l'industrie et l'informatique sont bien des réflexions de troisième ordre ; et nous avons appelé *systématicité* cette nouvelle modalité de la résolution de problème, quoique de manière intuitive et problématique à ce stade.

Nous avions déjà perçu, au chapitre précédent que, même si on parvenait à consigner l'ensemble d'un savoir pratique au sein d'un livre de procédure, l'usage de celui-ci requerrait à nouveau des règles, dont l'expression systématique au sein d'un algorithme pourrait conduire à dissoudre ce savoir dans un langage qui lui serait étranger, comme on l'a vu dans l'exemple des jeux de stratégie caractérisés par des situations « enchevêtrées ».

Or les études que nous avons consultées dans ce chapitre nous ont montré que de tels livres de procédures n'exprimeraient absolument pas l'essence du savoir technique, tel qu'on l'entend habituellement en milieu professionnel. Ce savoir est bien plutôt matérialisé par des *environnements techniques* que par des livres, qui organisent l'écologie cognitive des agents et leur permettent d'ancrer leur savoir pratique dans des habitudes patiemment construites. Les quelques règles et procédures explicites qui aident les personnes humaines à accomplir leurs tâches

ne sont pas des ersatz de procédures informatiques qu'il suffirait de rendre plus précises et plus rigoureuses, car elles ne servent pas en général à « mécaniser » le comportement humain, mais bien plutôt à le coordonner. Aussi la confrontation des personnes avec les systèmes informatiques est-elle souvent difficile. Elle met en évidence des logiques fondamentalement différentes d'assimilation du changement et d'adaptation du comportement. Systématique, la logique procédurale peut être obtuse, incapable de tirer parti de l'ambiguïté des situations, et surtout lourde à reconfigurer si la situation évolue d'une manière imprévue.

D'autre part cependant, on voit cette logique apparemment étrangère à nos savoirs pratiques se montrer d'une redoutable effectivité à résoudre les problèmes dont ils s'occupent. Nous avons suggéré que cette logique était la même que celle qui présidait à l'industrialisation des activités humaines, bien avant l'apparition des ordinateurs, celle d'une attitude réflexive visant à diviser le travail en tâches élémentaires standardisées afin de le recomposer de manière systématique, portant donc en elle le germe de cette *science des opérations* entrevue par Charles Babbage et Ada Lovelace. La programmation apparaît dans cette perspective comme la continuation et la généralisation de cette posture analytique, capable de prendre pour objet non seulement le travail productif, mais tout type de situation problématique qui peut être résolue par des opérations réglées. Nous avons dégagé trois formes d'avantages d'effectivité que l'informatique permet à cette réflexion de mettre en place : la conduite *automatique* d'opérations, la *contrainte* opérée sur les autres agents de la situation, et la *traçabilité*.

Il s'agit donc d'*une autre connaissance* qui est rendue possible par la programmation, étrangère aux savoirs pratiques immédiats des hommes, et pourtant construite, au moins initialement, à partir de leur analyse et de leur reconfiguration. Herbert Simon avait donc raison quand il entrevoyait la *conception* comme une faculté essentielle de l'homme pratique moderne ; mais il allait trop loin quand il en faisait la source de *tout* savoir pratique, en cela encouragé par sa propre hypothèse de la nature computationnelle de l'esprit. L'établissement de systèmes de règles pour diriger rigoureusement et exhaustivement l'action est une démarche hautement *artificielle*, s'il est permis de jouer sur une connotation différente de ce terme. Elle ne se met en place que dans certains cadres professionnels, notamment ceux où un changement d'échelle significatif est expérimenté.

De quelle nature est donc cette *autre connaissance* ? La notion de *systématicité*, en philosophie de la connaissance, est d'habitude associée à l'organisation

théorique du savoir et aux sciences. Cela voudrait-il dire qu'il existe une modalité *théorique* ou *scientifique* de la résolution de problème, de laquelle relèveraient la réflexion industrielle et informatique, par opposition aux modalités *pratique* et *technique* étudiées à la fin du chapitre précédent ? Une telle hypothèse est séduisante : La démarche industrielle, dont nous avons tant rapproché la programmation, ne trouve-t-elle pas son aboutissement dans ce qu'on appelé le *management scientifique* au début du xx^{e} siècle ?

Dans cette section, nous proposons une première exploration de cette hypothèse. Nous commençons (§ 91) par rapprocher les observations faites par Orr concernant le *besoin de comprendre* des techniciens d'une autre analyse, faite cette fois-ci dans le milieu même de l'informatique, par l'informaticien Peter Naur. Elle concerne une équipe de programmeurs qui, selon Naur, se montrait très effective dans la maintenance d'un logiciel grâce à sa *familiarité*, acquise sur la longue durée, avec son fonctionnement ainsi qu'avec la logique des problèmes auxquels il était préposé. Naur affirme de manière surprenante que ces programmeurs avaient « construit une théorie » de la situation.

Ce sens hétérodoxe de *théorie* est emprunté par Naur à Gilbert Ryle, et c'est donc à l'examen de la conception très originale de la connaissance théorique proposée par ce philosophe qu'est consacré l'essentiel de cette section. Nous résumons d'abord ses thèses (§ 92) avant d'en montrer le caractère problématique (§ 93).

§ 91. Le savoir pratique : une théorie de la situation ?

Nous avons tout à l'heure comparé la compréhension que cherchent à obtenir les techniciens de leurs photocopieurs à celle de l'ingénieur qui, parce qu'il conçoit la machine, est capable d'en décrire le mécanisme – et donc également ses défauts s'il y a un écart entre la conception et l'exemplaire concret. Cependant, le technicien de Xerox n'a pas accès à tout le savoir de l'ingénieur, mais doit dans une large mesure l'inférer des données partielles dont il dispose. Il se trouve donc dans une position assez similaire à celle d'un philosophe du Grand Siècle qui, observant les phénomènes naturels, en recherche les causes mécaniques c'est-à-dire, littéralement, leur explication sous le paradigme de machines construites par un grand Horloger ou un grand Mécanicien à partir des seuls principes élémentaires de la matière et du mouvement[101]. Dans notre cas, ce paradigme de l'investigation

101. Voir plus loin, section 7.4.

scientifique s'applique littéralement, puisque le département d'ingénierie de Xerox joue, pour les techniciens, le rôle du grand Horloger.

En d'autres termes, la posture des techniciens est essentiellement *théorique*, et on arrive au résultat inattendu que la forme supérieure du savoir permettant de résoudre les problèmes pratiques est le savoir théorique lui-même. Les procédures, que nous croyions avec Herbert Simon pouvoir fixer l'essentiel d'un savoir pratique original, se trouvent ici fermement placées dans son orbite.

On pourrait rétorquer que ce résultat n'est inattendu que pour qui rêvait à la possibilité de sciences de l'artificiel. La conception traditionnelle du savoir, qui remonte à Aristote, n'a-t-elle pas toujours affirmé que l'art imitait la nature ? Qu'y a-t-il donc d'étonnant à ce que le technicien, qui se trouve ici dans la position de l'artisan, doive « imiter » l'ingénieur pour rétablir le bon fonctionnement de la machine, de la même manière que le médecin doit comprendre le fonctionnement du corps humain lorsqu'il cherche à y rétablir l'ordre naturel[102] ?

Il nous est difficile à ce stade de prendre parti sur cette question. D'une certaine manière, *nous ne savons pas encore de quoi nous parlons* car, comme nous l'avons dit en introduction, ce qu'est le savoir théorique n'a pas été encore éclairci. Pour montrer ce caractère non évident, nous allons considérer une analyse qui prend également son origine dans le monde de la maintenance industrielle et qui, s'appuyant sur la réflexion de Gilbert Ryle, interprète de manière originale cette dimension « théorique » du savoir des techniciens.

L'analyse est due à Peter Naur, un des pionniers de l'informatique (il fut l'un des concepteurs principaux du langage ALGOL 60). De manière intéressante pour notre propos, l'analyse concerne l'évolution et la maintenance d'un logiciel industriel complexe, fortement personnalisé au cas particulier de chaque client. Une équipe centrale est chargée de l'installation des nouvelles versions du logiciel chez les clients et de la correction des erreurs. Cette équipe est en place depuis la programmation initiale et bénéficie d'une longue stabilité sur plusieurs années. De ce fait, elle est capable de répondre efficacement à tous les types de problèmes, notamment ceux survenant lors d'adaptations personnalisées, ou lors de nouvelles versions, en faisant appel aux seules connaissances de ses membres. Ceux-ci ont seulement besoin de se référer au code source du programme, sans besoin de consulter aucune autre documentation. Les autres équipes de soutien au contraire, même munies de toute la documentation disponible sur le logiciel, sont souvent dému-

102. Voir plus loin notre développement § 96.

nies face aux problèmes rencontrés[103].

Cet exemple ressemble beaucoup au précédent, et Naur en tire des conclusions similaires à celles d'Orr. Les livres de procédures sont des pis-allers dans la résolution de problèmes, et ils sont souvent inefficaces, même en milieu industriel – et même quand il s'agit de la maintenance de programmes, qui ne sont rien d'autre, à première vue, que des procédures ! Pour Naur, la croyance en l'efficacité des livres de procédures procède de l'illusion, qui règne selon lui dans les milieux informatiques, que le travail de la programmation peut être lui-même industrialisé :

> Une grande partie des discussions actuelles sur la programmation semble supposer qu'on peut l'assimiler à la production industrielle, le programmeur étant considéré comme une composante de cette production, qui doit être contrôlée par des règles de procédure et qui peut être facilement remplacée. Un autre point de vue connexe est que les êtres humains fonctionnent mieux s'ils agissent comme des machines, en suivant des règles. Un tel point de vue insiste donc sur les modes formels d'expression, qui permettent de formuler certains arguments en termes de règles de manipulation formelle. De tels points de vue s'accordent bien avec la notion, apparemment courante chez les personnes travaillant en informatique, selon laquelle l'esprit humain fonctionne comme un ordinateur. Au niveau de la gestion industrielle, ces points de vue amènent à traiter les programmeurs comme des travailleurs d'assez faible responsabilité, n'ayant besoin que d'une brève formation[104].

Trois thèses que nous avons déjà rencontrées sont associées ici. La première est un parti-pris en faveur de l'approche industrielle de la programmation dans le cadre du débat entre informaticiens sur la nature de leur profession[105] (« le programmeur [est] une composante de cette production »). La seconde est l'interprétation formelle de la programmation[106] (« l'insistance sur les modes formels d'expression »). Naur, initialement un collègue proche de Dijkstra (ils sont tous deux hollandais), se démarqua fortement de lui sur ces questions. La troisième n'est rien d'autre que l'hypothèse computationnelle de l'esprit, dont Herbert Simon était le promoteur (« l'esprit humain fonctionne comme un ordinateur »). En d'autres termes, toutes ces thèses relèveraient d'une même erreur fondamentale, que programmer serait une activité elle-même représentable par un programme.

Cette erreur n'est possible que si on restreint la programmation au codage de

103. (Naur 1985, p. 254).
104. (ibid., p. 260).
105. Voir plus haut, § 34.
106. Voir plus haut, § 47.

procédures selon des spécifications, et si on ne voit pas qu'elle concerne d'abord l'invention des spécifications elles-mêmes. Au contraire, cette invention doit être fondée dans une compréhension profonde du problème concret qu'il s'agit de résoudre, et que Naur assimile à la *construction d'une théorie* (*programming as theory-building*) :

> Le programmeur ayant une théorie concernant un programme peut expliquer comment cette solution se rapporte aux affaires du monde qu'elle aide à gérer. Une telle explication devra porter sur la manière dont les affaires du monde, tant dans leurs caractéristiques générales que dans leurs détails, sont, en un certain sens, inscrites dans le texte du programme et dans toute documentation additionnelle[107].

C'est cette compréhension du *fonctionnement* du programme qui lui permet d'assurer sa maintenance, tout comme les techniciens de Xerox avaient besoin de se représenter les mécanismes des photocopieurs afin de pouvoir les réparer :

> Le programmeur qui a la théorie d'un programme est capable de répondre de manière constructive à toute demande de modification à son propos, de manière à l'adapter aux affaires du monde auxquelles il contribue. La conception de la meilleure façon d'intégrer une modification à un programme établi dépend de la perception de la similarité entre la nouvelle demande et les possibilités opérationnelles déjà intégrées au programme. Cette évaluation doit être comprise comme comparant entre eux des aspects du monde. Elle ne peut avoir de sens que pour l'agent qui a cette connaissance du monde, c'est-à-dire pour le programmeur, et elle ne peut être réduite à un ensemble limité de critères ou de règles[108].

Il peut sembler surprenant que Naur assimile la programmation à une activité théorique, puisqu'il s'oppose à la position formaliste de Dijkstra qui cherchait à assimiler la programmation aux mathématiques[109]. On se serait plutôt attendu à ce qu'il défende, comme Simon, l'autonomie du savoir pratique comme *design* contre sa réduction possible à des principes théoriques.

Mais la position de Naur est plus subtile, voire plus subversive, puisqu'elle s'appuie sur la conception du savoir théorique promue par Gilbert Ryle, qui inverse son rapport de primauté vis-à-vis du savoir-faire (*know-how*), celui-ci étant la forme primordiale de tout savoir. C'est à cet exposé et à sa critique que nous consacrons la fin de ce chapitre.

107. (Naur 1985, p. 256).
108. (ibid., p. 256).
109. Voir plus haut, § 12.

§ 92. Les plans et les théories de Ryle

Pour Ryle, disposer d'une théorie est d'abord une compétence ; son texte même, ou l'ensemble de propositions qu'elle recouvre, en sont des productions, et non l'inverse. Ce n'est pas parce qu'on se représente les propositions de la théorie qu'on en acquiert la compétence – cela ne suffit en général pas ; mais au contraire parce qu'on dispose de la théorie qu'on sait également l'expliquer et la transmettre sous forme propositionnelle. Mais une théorie qui ne servirait qu'à être expliquée tournerait à vide. Elle doit également pouvoir être *appliquée*, s'insérer dans des pratiques autres que théoriques, « équiper l'esprit » de compétences supplémentaires[110]. Comme toute compétence, une théorie doit être acquise, soit par l'apprentissage, soit par l'investigation, au prix d'un effort, qui implique parfois l'assimilation patiente de certaines propositions, mais également la réflexion, l'exercice, l'exploration.

Cette interprétation ne restreint pas le terme *théorie* aux seules disciplines mathématiques et scientifiques, mais lui fait recouvrir tout type de réflexion discursive, comme le montre cet exemple de Ryle :

> La femme au foyer essayant de savoir si un tapis conviendra à un sol est engagée dans une modeste tâche de théorisation. Elle enquête sur quelque chose et les résultats de ses enquêtes pourront être explicités. Ce qu'elle explique à son mari et ce qu'elle fait avec le tapis exhibent la théorie à laquelle elle est parvenue[111].

Peu après, Ryle donne une deuxième définition de ce qu'est une théorie :

> J'utilise également le mot *théorie* pour couvrir les résultats de toute investigation systématique, que ces résultats constituent ou non un système déductif. Le récit par un historien du cours d'une bataille en est sa théorie[112].

Ryle n'explique pas en quel sens cette deuxième définition est complémentaire de la première. On peut voir qu'elle a une portée restrictive, qui entend se rapprocher de l'usage courant du terme concernant les théories scientifiques. Ryle prend soin cependant de conserver une certaine distance vis-à-vis de ce sens trop étroit, en ajoutant immédiatement qu'il n'entend pas nécessairement par là un *système déductif*, et en donnant le contre-exemple d'une investigation historique qui reconstruit le cours des événements, et que Ryle qualifie également de *théorie*.

110. (Ryle [1949] 2002, p. 286-287, 310).
111. (ibid., p. 288). Il est malheureusement caractéristique de ses préjugés que Ryle ait associé une théorisation « modeste » à la figure de la femme au foyer.
112. (ibid., p. 289).

Quelle est donc cette attitude systématique de l'investigation qui *fait* la théorie ? Comme souvent chez Ryle, le substantif *système* ne peut nous aider à comprendre l'adjectif *systématique* ou l'adverbe *systématiquement*. Le premier est utilisé dans *the Concept of Mind* de manière presque exclusivement péjorative, pour dénoncer l'illusion intellectualiste (le « système déterministe » des mécanistes, p. 20, le « système causal » des cartésiens, p. 66, le « système » des moralistes, p. 67). C'est au contraire l'attitude systématique de la recherche qui est première, et le système qui en est, éventuellement, le résultat hypostasié et nécessairement imparfait. Quant au terme *systématiquement*, tout ce qu'on peut glaner de ses occurrences dans le texte est qu'il s'oppose à *aléatoirement* (*haphazardly*) (p. 151). Il nous faut donc entrer plus avant dans la dynamique interne de l'effort théorique afin d'éclairer ce que Ryle peut vouloir signifier par là.

Ryle consacre tout le chapitre 9 de *the Concept of Mind* à la description de cette dynamique, que nous pouvons résumer par les points suivants :

— Élaborer (*to build*) la théorie est un effort qui doit être clairement distingué de la compétence qu'il permet d'acquérir, et qui permet d'utiliser (*to use*) la théorie[113]. Les opérations intellectuelles n'y sont pas les mêmes : déduire, par exemple, n'y signifie pas la même chose.

— Toute compétence théorique comprend une dimension didactique (pouvoir expliquer ou exposer la théorie) et une dimension applicative – pouvoir appliquer la théorie pour des fins autres que son exposition didactique, quelles qu'elles soient. Chacune de ces deux dimensions est nécessaire pour qu'on puisse parler d'une compétence théorique.

— La caractéristique du discours didactique théorique est sa forme propositionnelle, qui le distingue des autres modes de la communication – la conversation, la prière, l'ordre, etc. Cette forme est *impersonnelle* et *a-topique*, c'est-à-dire qu'elle structure une proposition de manière à ce qu'elle puisse être dite par quiconque, en une occasion quelconque. Une leçon théorique peut ainsi être interrompue, reprise plus tard, répétée, réexpliquée par un autre professeur, etc.[114]

— Grâce à sa dimension didactique, la réflexion théorique est *progressive*, selon les termes de Ryle, c'est-à-dire ouverte au progrès, ou féconde. Lorsqu'on utilise une théorie, on peut à la fois être maître et élève, s'adresser à soi-même

113. (ibid., p. 289).
114. (ibid., p. 311).

sa partie didactique, par exemple en en reprenant les principes, ou les théorèmes, ou leurs démonstrations. « On s'entraîne soi-même à dire et à faire des choses qui ne sont pas de simples échos des mots par lesquels l'entraînement est donné » et, de la même manière qu'on peut se donner à soi-même des ordres, « on peut se dire des choses dont on tire ensuite parti dans de nouveaux coups didactiques[115] ». Ces « coups » ne sont pas nécessairement des déductions logiques, il peut s'agir de calculs, d'une synthèse historique, d'une traduction, etc.

— Le caractère impersonnel du discours théorique assure enfin que les progrès jugés intéressants peuvent lui être intégrés, et qu'il est donc cumulatif : « Nous pouvons ainsi apprendre ce que nos grands-parents enseignaient à nos parents, et ce que nos parents modifièrent ou ajoutèrent à ces leçons […] Les sciences croissent parce que le jeune étudiant peut, s'il a une scolarité appropriée, débuter là où Euclide, Harvey et Newton s'arrêtèrent[116] ».

On voit ici la forte cohérence de la conception ryléenne : tous ces points, pris ensemble, décrivent un *mouvement* interne de l'effort théorique, qui incessamment alterne entre acquisition et croissance du savoir d'une part, et sa récapitulation dans son moment didactique d'autre part, où, énoncé sous une forme propositionnelle a-topique, il peut être transmis et croître à nouveau. Là réside la vertu de cette forme propositionnelle, du *know-that* que les philosophes ont longtemps confondu avec la connaissance tout court : permettre à la connaissance proprement dite, qui est avant tout compétence et activité, d'entrer dans une dynamique explosive de cumul et de transmission. Par là s'explique également la nature de l'activité scientifique : il s'agit d'un effet théorique systématique qui se préoccupe seulement de son développement didactique, repoussant indéfiniment hors de son champ l'examen de ses conséquences pratiques – ce qui ne veut pas dire qu'elles n'existent pas. Elles sont comme l'étudiant qui ne quitte jamais l'Université, parce qu'il devient professeur : il s'agit là d'une carrière possible ouverte par les études universitaires, mais elle est minoritaire.

Le caractère systématique de l'investigation se trouve dans deux caractéristiques de l'effort théorique, tel que nous venons de le décrire. La première est la séparation claire entre le moment d'élaboration de la théorie et le moment de son utilisation. On trouve que le résultat d'une investigation est systématique lors-

115. (Ryle [1949] 2002, p. 312)
116. (ibid., p. 311).

qu'on ne peut, *à l'usage*, lui ajouter ou retrancher quoi que ce soit. Tout a été prévu par avance ; dans quelque situation qu'on l'éprouve, elle fournira aisément une solution. Cette facilité doit être opposée à la difficulté préalable qu'il y a eu à *construire* la théorie et à s'assurer de sa robustesse à tous points de vue. Ryle écrit, dans son style colorié caractéristique :

> Lorsqu'un agriculteur a tracé un chemin, il est capable de le parcourir facilement en tous sens. C'est pour cela que le chemin a été fait. Cependant le travail de tracer le chemin n'était pas une promenade, mais un labeur où il fallut marquer le sol, creuser, ramasser le gravier, rouler les pierres et drainer. […] De même, une personne qui possède une théorie peut, entre autres, l'exposer à soi-même ou au monde, tout entière ou chacune de ses parties ; elle peut, pour ainsi dire, se promener en prose de n'importe quelle partie de la théorie à n'importe quelle autre. Mais le travail de construction de la théorie consistait à tracer des chemins là où il n'y en avait pas encore[117].

On voit affleurer dans ce texte, qui ne cite pas le terme, l'un des sens de *système*, comme ensemble dont les parties se répondent les unes aux autres, de manière à former un tout organisé. La systématicité est, dans cette image, l'aménagement d'un lieu de telle sorte qu'en chaque point tous les autres soient immédiatement accessibles.

La seconde caractéristique d'une théorie est son indépendance au contexte. Quand Ryle insiste sur l'impersonnalité et l'a-topicalité du discours théorique, il ne s'intéresse pas seulement à l'écorce stylistique des propositions. Cette forme ne fait que refléter un effort plus profond de dé-contextualisation. Toute théorie, dans son effort systématique, cherche à expliciter ses propres conditions de validité, et donc à définir ses termes, les situations auxquelles elle s'applique, les hypothèses qu'elle requiert. Poussé à son terme, cet effort aboutit aux systèmes formels qui distinguent radicalement syntaxe et sémantique, la forme et la matière du raisonnement. Les signes propositionnels ne sont plus alors que des symboles sur lesquels peut se déployer le calcul, c'est-à-dire une application de règles univoques et vérifiables par tous. Un système formel est systématique, en ce qu'il exclut explicitement les cas qu'il n'a pas prévus, et qu'il rend possible une combinatoire maximale de possibilités pour ceux qu'il a intégrés.

On comprend pourquoi les deux définitions de *théorie*, données plus haut, peuvent se rejoindre. Quand la femme au foyer de Ryle entreprend de résoudre son problème « systématiquement », elle se trouve bien engagée dans une forme

117. (ibid., p. 289).

spéciale d'activité théorique. Cette activité, que Ryle juge similaire à celle de l'enquête théorique des sciences, est la *planification*. Ce terme joue un rôle central dans *the Concept of Mind*. C'est ainsi l'expérience fréquente de l'action planifiée qui convainc les adversaires intellectualistes de Ryle que toute action doit être conçue sous ce modèle, c'est-à-dire être précédée par la représentation d'une règle ou d'un raisonnement. C'est ce même concept qu'utilise Ryle pour réfuter cette illusion, grâce à un argument de régression à l'infini : car s'il faut toujours un plan avant d'agir de manière rationnelle, l'acte de planifier lui-même en requerra un, et ainsi de suite[118]. Enfin et surtout, de manière récurrente (12 fois), et en particulier au chapitre 9, Ryle associe étroitement les termes *plan* et *théorie*, comme dans les expressions suivantes :

> Une personne engagée dans une tâche de *planification* ou de *théorisation*... (p. 263)
> Comme les opérations de *planification* et de *théorisation* peuvent elles-mêmes être caractérisées comme motivées...(p. 280)
> Nous ne considérons pas la conversation comme une *théorisation* ou une *planification* de bas niveau...(p. 283)
> Avoir une *théorie* ou un *plan* n'est pas en soi faire ou dire quoi que ce soit...(p. 286)
> À la différence de certaines formes de *théories* ou de *plans*...(p. 301)
> etc.

La planification et l'investigation théorique sont liées en ce qu'elles présentent toutes deux les mêmes caractères de systématicité : d'une part, la séparation nette entre le moment de construction du plan ou de la théorie et celui de leur usage ; et d'autre part, la recherche d'une validité « universelle », c'est-à-dire qui s'élève au-dessus des variations de leurs conditions et de leurs circonstances d'usage.

Ainsi, par exemple, l'ingénierie se distingue essentiellement des réflexions pratiques et techniques normales en ce que celles-ci, si elles peuvent certes faire appel, de temps à autre, à la posture théorique, ne la requièrent pas nécessairement. Un maître maçon peut construire un pont en s'inspirant des modèles qu'il a assimilé et de son expérience qui lui permet de bien estimer les portances et les quantités de matière nécessaires ; mais il ne fait pas là œuvre d'ingénieur. Au contraire ce dernier entreprend une démarche systématique de conception *avant* d'entreprendre la construction. L'explicitation de ses données de départ et de ses hypothèses (par exemple le tracé du lit de la rivière, la force des courants, la vitesse maximale des vents) lui garantit une forme d'indépendance au contexte : on ne construit

118. (Ryle [1949] 2002, p. 30-31, voir aussi p. 176-177).

pas un pont en été sans penser aux conditions hivernales, et même aux pires qu'il est raisonnable d'imaginer, au trafic routier le plus exigeant, à l'éventualité d'un incendie, etc. L'ingénieur est donc d'emblée dans une démarche de planification systématique.

C'est cette démarche qui explique la proximité de l'ingénierie avec les sciences, dont elle est pourtant radicalement distincte par l'intention. La rapprochant de la stratégie militaire et des jeux, Ryle écrit :

> La connaissance des échecs ou du bridge est un accomplissement intellectuel qui s'exerce en tentant de gagner des parties ; la stratégie, un autre en tentant de gagner des batailles et des campagnes ; La scolarité de l'ingénieur et son expérience de terrain lui apprennent à concevoir des ponts et non, sinon *per accidens*, à élaborer et exposer des théories.
>
> La raison pour laquelle nous appelons ces jeux et ces travaux « intellectuels » n'est pas difficile à trouver. Non seulement l'éducation nécessaire à la maîtrise, mais aussi bon nombre des opérations nécessaires à leur pratique sont homogènes avec celles requises pour et dans les tâches d'élaboration, d'exposé et d'application des théories. […] L'ingénierie ne fait pas progresser la physique, la chimie ou l'économie ; mais la compétence en ingénierie n'est pas compatible avec une innocence complète quant à ces branches théoriques[119].

L'ingénierie n'est pas théorique en ce qu'elle mobilise les savoirs des sciences, c'est-à-dire en ce que l'ingénieur appliquerait les formules qu'il a apprises d'elles pour concevoir le pont. Ce fait est bien plutôt une conséquence de sa démarche. Si la science lui propose des modèles tous préparés pour la résolution *systématique* de toutes les conditions de son problème, c'est qu'elle avait cet objectif en vue. Réciproquement, dans son enquête systématique l'ingénieur peut être amené à poser de nouvelles questions aux sciences, voire à susciter de nouvelles recherches dans leurs domaines. La démarche de l'ingénieur n'est pas fondamentalement asservie à celle de la science, mais il la rencontre dans sa recherche systématique d'optimisation et de vérification, car elle-même vise une telle classe de problèmes.

D'une manière générale, la démarche de systématicité dans laquelle sont engagées les pratiques comme l'ingénierie ou la programmation ne signifie donc pas qu'elles pourraient être codifiées en des procédures rigides, bien au contraire. On est systématique parce qu'on a *construit* ou *assimilé* une théorie, et non parce qu'on l'applique. La systématicité n'est rien d'autre qu'une propriété des *compétences* que les praticiens de ces disciplines doivent acquérir.

119. (ibid., p. 315).

Cela montre également que les pratiques ne dépendent pas d'une élaboration théorique, mais la précèdent :

> La pratique efficace précède sa propre théorie ; les méthodologies présupposent l'application des méthodes et de l'investigation critique dont elles sont les produits. C'est parce qu'Aristote se vit, lui-même et d'autres, parfois raisonner intelligemment, et parfois stupidement [...] qu'[il] fut capable de donner à ses élèves les maximes et les prescriptions [de la logique]. Il est donc possible aux gens de réaliser certains types d'opérations intelligemment, alors qu'ils ne sont pas encore capables de considérer aucune proposition prescrivant comment elles devraient être réalisées[120].

C'est donc dans cette perspective ryléenne radicale qu'il faut comprendre l'interprétation par Naur de la programmation comme étant une activité théorique. Il n'est pas sûr que Ryle lui-même aurait accepté cette expression, sinon en un sens métaphorique où l'ingénieur aurait également *une théorie de son pont*. Elle s'accorde en tous cas profondément à la notion d'une discipline de *planification systématique* : 1) La « division du travail » entre réflexion préalable et action est inscrite dans la distinction essentielle entre le texte du programme et le processus informatique qui le réalise, entre la phase de conception et la phase d'exécution. 2) L'indépendance au contexte s'y trouve également par l'idée d'une séparation stricte entre le domaine de la machine et celui de la situation, qui représente toutes les circonstances auxquelles le programme devra savoir réagir. Comme nous l'avons vu en effet[121], une spécification est toujours biface, et concerne tout autant ce que doit réaliser la machine que le *savoir* que nous possédons concernant la situation. C'est en ce sens que la spécification contient bien, comme le dit Naur, une *théorie de la situation*.

À ce titre, on pourrait aller jusqu'à avancer que la programmation apparaît, par rapport à l'ingénierie et aux autres pratiques intellectuelles, comme la *forme* même de la planification systématique. Nos analyses du chapitre 3 nous permettent de la voir tout entière dans ce double geste de séparation « *de jure* » entre conception et exécution d'une part, et entre machine et situation d'autre part, si bien qu'elle peut être appliquée à toute réflexion technique engagée dans une telle démarche de systématicité.

120. (Ryle [1949] 2002, p. 30).
121. Voir plus haut, § 57.

§ 93. Insuffisances des suggestions de Ryle

Nous sommes partis, au début du chapitre précédent, de l'hypothèse de Simon, que la programmation pouvait être la forme générale de la *conception* et de la réflexion technique. Malgré l'attractivité d'une telle thèse, et même en la corrigeant des biais propres à Simon, nous avons dû admettre, dans le cours de ce chapitre, que son exigence de systématicité la rendait étrangère, voire opposée, aux formes usuelles de la réflexion pratique et technique des hommes. Nous avons plutôt mis en évidence son étroite parenté avec la réflexion industrielle, dont l'association avec la démarche scientifique est récurrente depuis le début du XIX^e^ siècle. À partir de là, une hypothèse inverse s'est dégagée : la programmation n'est-elle pas le signe d'une réflexion *scientifique* sur la pratique, et ne relève-t-elle pas donc de l'esprit théorique qui analyse *systématiquement* un problème avant d'en proposer une solution ? La préférence des techniciens observés par Orr pour la compréhension du mécanisme sur l'application des procédures nous permettait en effet de voir, plus généralement, que la réflexion pratique se faisait naturellement théorique, et que l'explicitation technique des procédures n'était peut-être qu'un pis-aller, ou un stade intermédiaire de la connaissance. Cette hypothèse rejoint d'ailleurs l'antique idée que la technique, d'un point de vue épistémologique, n'est qu'une étape vers la science[122].

Cependant la référence à la *science* et à la *théorie* pour expliquer ce que serait la résolution *systématique* de problème est-elle vraiment éclairante ? Il est permis d'en douter, puisque la programmation, aussi bien que l'industrie, ne semblent pas requérir de théories préalables pour résoudre leurs problèmes. Nous avons déjà noté[123] que la programmation était à la portée d'un enfant de 7 ans ; sous-entendre que celui-ci disposerait donc d'une « théorie » semble étrange. On pourrait dire : *le langage de programmation est cette théorie*. Une telle assertion n'est pas fausse, mais elle ne cadre décidément pas avec le sens usuel qui est donné à ce terme. Quant à l'industrie, on l'a vu, ni elle, ni même la technologie, ne consistent en l'application des résultats de la science aux processus de production[124]. Il semble au contraire, comme l'étude de Ryle nous l'a suggéré, que c'est parce qu'elles relèvent de la même démarche que la science qu'elles peuvent la mobiliser lorsque nécessaire.

122. Voir plus loin, § 96
123. Voir plus haut, § 4.
124. Voir plus haut, § 86, et également § 15.

On dira : c'est bien la *méthode* ou la *démarche* scientifique qui est importante ici. Mais alors, quelles caractéristiques de cette méthode sont ici importantes ? La formulation d'hypothèses, l'appel à l'expérimentation, la falsifiabilité, etc. ces éléments qui viennent naturellement à l'esprit quand on évoque la démarche scientifique moderne, ne semblent pas du tout centraux dans la démarche de la programmation et de l'industrie. C'est au contraire l'idée d'une *approche systématique* de la chose à traiter qui semble être leur point commun, et on est ainsi revenu à notre point de départ.

C'est d'ailleurs à l'idée de *plans systématiques* plutôt que de *théories* que semble le mieux s'accorder l'idée de *procédure* que nous cherchons à élucider, mais Ryle n'explique pas quelle différence profonde il fait entre *plans* et *théories* qu'il accole si fréquemment, ni pourquoi le savoir théorique ne s'exprime pas, finalement, sous la forme prescriptive d'un plan, mais sous la forme de propositions descriptives simples, d'où il tire d'ailleurs l'expression *knowing-that* sans l'expliquer.

Peut-être Ryle cherche-t-il justement à suggérer la continuité fondamentale entre toutes ces formes de savoir discursif, ou en tous cas leur « air de famille », qu'on ne saurait fixer dans des typologies tranchées ? Mais cela signifie-t-il qu'il y a continuité, ou convertibilité entre les régimes prescriptif et descriptif du savoir, entre *is* et *ought* ?

Aussi semble-t-il que la caractérisation de la systématicité par Ryle, que nous avons tenté de résumer par l'opposition entre construction et usage, et par l'impersonnalité de la forme propositionnelle, n'est pas suffisante. Or c'est cela même qui nous avait semblé central pour la compréhension de la programmation.

Toutes ces questions ne sauraient être résolues par la seule exégèse des textes de Ryle. Il a trop peu écrit sur la technique, sur le savoir scientifique et sur les mathématiques pour qu'on puisse espérer prolonger sa réflexion en direction de notre questionnement. Cependant, il met en évidence un fait d'importance, que notre réflexion vacille dès qu'elle tente de prendre appui sur les concepts de *théorie, pratique, machine, règle, système, etc.* que nous avons hérités de la tradition, et cela sans doute parce qu'elle tente de les utiliser pour décrire un phénomène épistémique (la programmation) qui n'avait pas encore cours lorsque leurs sens modernes se cristallisèrent. Aussi, c'est à leur archéologie que nous sommes à présent invités. Bien sûr, il ne s'agit pas là de débuter un vaste chantier de fouilles, mais de creuser seulement *autour* du lieu qui nous intéresse, celui marqué par

l'idée d'une élaboration de procédures systématiques pour la direction de l'action et des machines.

Troisième partie

Théorie et pratique

Chapitre 6

Technique et investigation (Aristote)

§ 94. Introduction de la troisième partie

Nous continuons dans cette partie l'exploration de la *forme de connaissance* que serait la programmation. Si elle est une résolution de problème qui se fait à l'aide d'une machine, il nous reste toujours à élucider ce qu'est résoudre un problème, et en quel sens nous entendons le terme *machine*.

La partie précédente nous a enseigné deux choses principales concernant le premier point. D'abord, la résolution de problème ne semble pas pouvoir circonscrire un domaine de connaissance, aussi large soit-il. D'un certain point de vue, il semble naturel de la rapprocher de la connaissance pratique, qui délibère afin de transformer des situations problématiques en situations satisfaisantes. C'est ainsi que l'entend Herbert Simon lorsqu'il en fait l'essence des *professions* (droit, ingénierie, médecine, etc.). D'un autre point de vue cependant, les mathématiques et les sciences résolvent également des problèmes, qui cependant ne semblent pas transformer des situations mais plutôt des significations.

Une telle expression (« transformer des significations ») peut néanmoins sembler métaphorique, et on peut douter qu'il y ait rien d'autre qu'une simple analogie dans un tel rapprochement. Constituer la *résolution de problème* comme objet épistémologique *unique* à travers des usages si diversifiés, et notamment par-delà la distinction entre connaissance théorique et pratique, n'est donc pas un geste évident. Cependant, c'est bien vers cette possibilité d'unification que semble pointer la programmation, par son effectivité à résoudre une grande quantité de problèmes, quels que soient leur niveau de complexité (faible ou élevée) et leur domaine (scientifique, technique ou autre) – et par l'idée qu'il serait possible d'avoir

des *langages universels* de description et de résolution de problèmes[1].

Cette question est rendue plus complexe encore par la seconde chose que nous a enseignée la partie précédente. La programmation, comme les méthodes industrielles, semble relever d'une *modalité* très spécifique de la résolution de problème, que nous avons nommée *systématique*, par opposition à ses modalités *pratique* et *technique* dont nous avons clarifié le sens. Or l'idée de *systématicité* semble plutôt caractéristique, quand il est affaire de connaissance, d'une posture théorique ou scientifique. Ce rapprochement semble également lier les notions de *machine* et de *système* en un rapport étroit, puisque c'est l'emploi de machines qui nous a semblé caractéristique de la forme de résolution de problème qu'est la programmation, et que sont également, dans une à peine moindre mesure, les méthodes industrielles.

Les questions directrices de cette partie sont donc les suivantes :

1. En quelle mesure doit-on distinguer les disciplines théoriques et pratiques de la connaissance ? en particulier, est-il légitime de parler de manière unitaire de *résolution de problème* à travers toutes ces activités ?
2. Que signifient les termes *machine* et *système*, et en quoi permettent-ils l'émergence d'une posture *systématique* de la résolution de problème ?

Il est important de bien voir que ces deux questions, l'une portant sur le genre, et l'autre sur la différence spécifique de la programmation, ne sont pas de simples exercices scolastiques. Cette différence spécifique n'est pas un accident qui advient comme de l'extérieur à son genre, à l'occasion, par exemple, d'une découverte technique. Cette différence spécifique, que la programmation a en commun avec l'approche industrielle du travail humain, est la possibilité, semble-t-il interne, *simplement logique*, qu'a la résolution de problème *de prendre une forme systématique*. Mener cette enquête, qui peut paraître nous éloigner de la programmation, est nécessaire pour comprendre l'ampleur des bouleversements qu'elle occasionne dans tous les champs de notre monde humain : c'est qu'elle est la reconfiguration intime et achevée d'une de nos manières essentielles de penser.

Ces deux questions portent, d'une certaine manière, sur la distinction courante qu'on fait entre connaissance théorique et connaissance pratique. Deux sens de cette distinction – parmi bien d'autres possibles – semblent en effet essentiels pour notre réflexion. D'une part, il y a la connaissance (théorique) de *ce qui est* et celle (pratique) de *ce qu'il faut faire*. Cette distinction est la même que celle désignée

1. Voir plus haut, § 11.

par Herbert Simon selon les termes *is* / *ought*[2]. Penser la résolution de problème comme une forme *unique* de connaissance requiert qu'on se convainc d'abord d'avoir affaire là à la même notion de connaissance, par-delà cette distinction, ce qui est rien moins qu'évident pour un très large pan de la tradition philosophique, comme on va le voir.

D'autre part, il y a une *modalité* pratique de connaître, et une autre théorique. La première passe par la réflexion libre concernant le problème à résoudre, et qu'on appelle en philosophie, suivant Aristote, *délibération* ou *investigation* ; elle peut se référer à des règles et à des artefacts cognitifs préparés à l'avance pour la guider, et dans ce cas on l'appelle technique. La seconde modalité passe par la mobilisation d'une théorie ou d'une doctrine pour éclairer le problème à résoudre, et c'est pour cela qu'on la présente souvent comme une *application* ou comme une *déduction* ; on la qualifie souvent de *systématique*. Cette distinction, aussi floue soit-elle, semble au contraire essentielle pour définir la programmation.

Le problème est que ces deux sens sont étroitement associés depuis Aristote. Comme nous allons le voir, le Stagirite, à qui l'on doit cette distinction entre deux formes de connaissances, associe naturellement la *modalité* pratique de connaître, la délibération, à la considération de *ce qu'il faut faire*, et sa modalité systématique à la considération de *ce qui est*. Chercher à savoir *systématiquement ce qu'il faudrait faire*, même dans un cadre technique, semblerait à Aristote incongru, sinon impossible. Nous avons d'ailleurs nous-même récusé l'idée que la programmation et les méthodes industrielles seraient de simples *applications* de théories préconçues.

Cette partie est consacrée à tenter de « délier tous ces nœuds », c'est-à-dire de comprendre l'importance de ces distinctions, et d'évaluer si elles sont toujours pertinentes. Nous avons choisi de le faire en suivant l'histoire de la philosophie de la connaissance. La justification en est double. D'une part, il nous semble impossible de manipuler des notions aussi lourdes d'histoire et de tensions sans tenter d'en faire, *a minima*, la généalogie conceptuelle, au moins dans les dimensions qui nous intéressent. La considération des idées à l'étape de leur formation, où elles sont encore rudimentaires, malléables, et où elles apparaissent dans toute l'ouverture de leurs développements possibles, nous semble aussi bien féconde pour la réflexion que requise pour éviter la naïveté du philosophe qui traiterait des concepts de *machine, système, règle, procédure, théorie, pratique, etc.* comme s'il s'agissait

2. Voir plus haut, § 66.

d'évidences intemporelles.

D'autre part, comme il s'agit de comprendre à présent les *conditions de possibilité* de la programmation, il nous a semblé nécessaire d'aborder ces questions telles qu'elles pouvaient se présenter *avant* l'apparition des ordinateurs, et même, vu le lien que nous avons relevé avec l'industrie, avant le XIX^e^ siècle – avec une attention particulière aux Temps modernes (1500 - 1800) où se forment progressivement et difficilement nos concepts actuels de *machine* et *système*. Ce sujet occupe l'essentiel du chapitre 7 central de cette partie. C'est là que nous pourrons le mieux apercevoir l'interdépendance de l'évolution des idées et des techniques dans la formation de cette nouvelle manière *systématique* de produire, d'agir et de penser.

Ce chapitre est encadré par une réflexion plus large concernant la distinction de la connaissance théorique et pratique : le chapitre 6 étudie la position initiale d'Aristote, qui nous a légué cette distinction, et le chapitre 8 la met en débat, à partir de sa remise en cause kantienne et de sa réhabilitation multiple au XX^e^ siècle. Comme nous espérons le montrer, les points en débat sont structurants pour appréhender la possibilité d'une planification systématique de l'action et donc de la programmation.

Nous ne prétendons pas dans cette partie faire de l'*histoire des idées*, ni même de l'histoire de la philosophie. Les chapitres centrés sur Aristote et Kant sont à cet égard des exercices tout à fait classiques d'interprétation de thèses soutenues par des philosophes du passé, dont nous présupposons qu'elles peuvent être pertinentes aujourd'hui encore pour éclairer notre réflexion. Si nous tentons d'être sensibles à leur contexte culturel, nous les discutons pied à pied, comme si Aristote et Kant venaient nous présenter leurs thèses *aujourd'hui même*, dotés du même savoir que nous.

Le chapitre 7, qui s'inspire d'observations sur le temps long, tente soigneusement d'éviter toute conjecture historique hasardeuse, que nous serions bien en peine de défendre. Nous nous appuyons certes sur un matériau historique important, le plus complet possible, que nous avons pu rassembler grâce aux travaux des historiens de la philosophie, des sciences et des techniques des Temps modernes. Dès que nous l'avons pu, nous avons tenté de vérifier nos analyses dans les sources primaires, mais la masse même de la matière examinée ne peut nous mettre à l'abri d'oublis importants et d'erreurs d'interprétation.

Cependant, si notre matériau est historique, notre visée, elle, ne l'est absolu-

ment pas. Il ne s'agit pas pour nous d'identifier par exemple des influences secrètes entre philosophies ou des présupposés communs à une certaine époque, mais plutôt de montrer comment le passage de la vision aristotélicienne de la technique à celle de Kant est conceptuellement possible, au prix de certains déplacements majeurs de sens concernant la notion de connaissance, où les idées de *machine* et *système* sont amenées à jouer un rôle central. De ce fait, nous ne mentionnons presque rien des facteurs culturels, économiques et politiques qui ont accompagné cette histoire conceptuelle, car là n'est ni notre question, ni notre compétence. Cela ne veut pas dire qu'ils ne furent pas déterminants.

Une dernière remarque concerne notre utilisation de l'expression *résolution de problème* dans cette partie. Nous l'emploierons par commodité, en suivant l'ensemble de ses usages courants aujourd'hui, voire en les élargissant quelque peu en direction des explications, qu'on rangerait plutôt sous le titre de réponses à des *questions*. Il s'agit pour nous de désigner, de manière problématique, l'ensemble dont nous cherchons à saisir l'unité conceptuelle. Il faut à ce titre remarquer que, si le terme *problème* existe et est abondamment utilisé dès l'Antiquité, son usage y est cependant circonscrit à la dialectique et à la géométrie, ce qui restera le cas jusqu'à la fin des Temps modernes[3]. La lectrice historienne trouvera donc cet emploi forcé. Nous lui demandons la patience : nous espérons que la légitimité de cet usage se dégagera progressivement, et nous reviendrons sur cette question en fin de partie.

§ 95. Introduction du chapitre

Comme nous l'avons dit, la tradition philosophique et, à travers elle, le sens commun, ont hérité d'Aristote la distinction entre connaissance théorique et connaissance pratique. A y regarder de plus près cependant, les explications qu'en donne le Stagirite n'ont plus rien d'évident pour nous. Que veut dire par exemple que la science (ἐπιστήμη) s'occuperait des « des êtres dont les principes sont nécessaires », tandis que les disciplines productives et pratiques porteraient sur ceux « qui peuvent être autrement »[4] ?

3. Voir par exemple l'article *Problème* dans la première édition du *Dictionnaire de l'Académie Française*, 1694.
4. *Éth. Nic.* VI.2, 1039a25. Les éditions consultées des textes d'Aristote se trouvent dans la bibliographie. Les textes grecs sont tirés du *Thesaurus Linguae Grecae*. Nous utiliserons les abbréviations suivantes : *Métaphysique* = *Métaphysique*, *Éthique à Nicomaque* = *Éthique à Nicomaque*, *Premiers analytiques* = *Premiers analytiques*, *Seconds analytiques* = *Seconds*

Aristote maintient par ailleurs, au sein de ce second domaine, une séparation également nette entre les disciplines productives, qui produisent des artefacts ou des résultats matériels : menuiserie, médecine, navigation, etc. et les disciplines pratiques, qui portent sur les actions humaines, comme l'Éthique et la Politique.

Cette séparation rigide de la théorie et de la pratique selon des domaines d'étude disjoints ne correspond plus à notre conception intuitive de la connaissance rationnelle. D'abord nous sommes habitués à postuler qu'il existe des fondements théoriques à toute connaissance technique, et réciproquement, à rechercher les applications pratiques de toute découverte théorique. Par ailleurs, le projet des sciences humaines a fait émerger la psychologie, la sociologie, l'économie, la politique, comme des disciplines théoriques à part entière, distinctes des préceptes pratiques qui pourraient en découler.

Cette remise en cause du modèle aristotélicien étend en droit le domaine de la connaissance théorique à l'ensemble du réel, et postule donc de toute technique et de toute pratique une fondation théorique possible. Cette thèse est l'œuvre de la philosophie moderne. Kant l'exprime parfaitement lorsqu'il propose de refonder la distinction entre les deux types de connaissance sur de toutes nouvelles bases. Si en effet les règles de la technique et les conseils de la prudence ne sont en droit que des « corollaires » de la science, la connaissance pratique, afin d'avoir un fondement autonome, ne doit comprendre que les enseignements de la loi morale :

> Lorsqu'on divise la philosophie, en tant qu'elle comprend des principes de la connaissance rationnelle des choses par concepts […] comme à l'habitude en philosophie théorique et en philosophie pratique, on procède tout à fait correctement. Mais alors les concepts, qui indiquent leur objet aux principes de cette connaissance rationnelle, doivent aussi être spécifiquement différents. […] Or il n'y a que deux espèces de concepts, qui admettent autant de principes différents de la possibilité de leurs objets : les *concepts de la nature* et le *concept de la liberté*. […] Or jusqu'à présent il s'est fait un grand abus de ces expressions pour la division des différents principes et avec eux de la philosophie : on tenait en effet pour identiques le pratique suivant les concepts naturels et le pratique suivant le concept de la liberté. […] [Mais] toutes les règles techniques-pratiques (c'est-à-dire celles de l'art et de l'habileté en général ou aussi de la prudence comme habileté à agir sur les hommes et sur leur volonté) ne doivent, dans la mesure où leurs principes reposent sur des concepts, être comptées que comme des corollaires de la philosophie théorique. […] Aussi peu donc que la solution des problèmes de la

analytiques, *Topiques* = *Topiques*, *Rhétorique* = *Rhétorique*, *Politique* = *Politique*.

> géométrie pure appartient à une partie particulière de celle-ci, ou que l'arpentage mérite le nom de géométrie pratique, différente de la géométrie pure, comme seconde partie de la géométrie en général, aussi peu et encore moins même l'art mécanique ou chimique des expériences ou des observations doit-il être considéré comme une partie pratique de la théorique de la nature, et enfin l'économie domestique, rurale, politique, l'art des relations sociales, les prescriptions de la diététique, même la doctrine générale du bonheur, pas même l'art de refréner et de dompter les passions à son profit, doivent-ils être comptés comme appartenant à la philosophie pratique ; […] c'est, en effet, que [ces disciplines] ne comprennent toutes que des règles de l'habileté, par conséquent seulement techniquement pratiques, pour produire un effet possible d'après les concepts naturels des causes et des effets, et que ces règles, puisqu'elles appartiennent à la philosophie théorique, sont soumises à ces prescriptions en tant que simples corollaires de celle-ci (de la science de la nature) et ne peuvent prétendre à occuper une place dans une philosophie particulière, dite pratique[5].

Nous avons cité ce texte célèbre en longueur car Kant y opère un déplacement radical des domaines de la technique et de la pratique. Alors qu'Aristote oppose aux sciences théoriques aussi bien les disciplines techniques que les disciplines pratiques, Kant associe connaissance théorique et connaissance technique en un seul vaste domaine portant sur la connaissance par concepts de la nature. Plus encore, il ampute la connaissance pratique de l'essentiel de la prudence aristotélicienne, qui mêle indissociablement la capacité à viser le bien et l'habileté à le réaliser. Et, afin de ne laisser aucune ambiguïté, aussi bien la « doctrine du bonheur » elle-même, qui était indissociable pour Aristote de la visée humaine du bien, que « l'art de réfréner les passions », si essentielle à la vertu selon le Stagirite, sont également embarqués dans cette relocalisation massive de la connaissance humaine dans le domaine « théorique », débarrassant ainsi le domaine pratique de toute autre considération que celle de la loi morale.

Ce chapitre commence par considérer la conception aristotélicienne de la technique. Comme on va le voir, son statut est un sujet toujours très actif du commentaire aristotélicien. Non pas que les textes soient obscurs ou contradictoires ; au contraire, Aristote se réfère souvent aux idées de connaissance théorique, technique et pratique, de manière globalement très cohérente, et ce à travers de nombreux textes (*Métaphysique, Éthiques, Analytiques, de l'Âme, Parties des Ani-*

5. *Critique de la faculté de juger*, Introduction, I, Akademie Ausgabe, t. V, p. 167 (abrégé par la suite en Ak. V, 167). Traduction française d'A. Philonenko, p. 21-22.

maux, etc.) qui renvoient les uns aux autres, explicitement ou implicitement. La difficulté rencontrée à la lecture des textes semble plutôt être celle même soulevée plus haut, qui est de comprendre la nature et l'ampleur de la distinction qu'Aristote établit entre théorie et pratique. Les deux textes majeurs qui exposent le sujet, *Mét.* A.1 et *Éth. Nic.* VI, sans être donc contradictoires, ont des perspectives et des inflexions différentes. Comme le dit par exemple Allen, « Aristote a du mal à mettre en avant quelles différences entre raison théorique et raison pratique il juge fondamentales »[6]. À la lecture des textes en effet, on ne trouve pas de différence majeure évidente entre science et technique, ni quant à leurs objets, ni quant à leurs modes de raisonnements. Ainsi, c'est le Stagirite lui-même qui semble ouvrir la voie à une interprétation « réductrice » de la technique à la théorie et donc, à la critique radicale formulée par Kant.

Dans les trois premières sections de ce chapitre, nous proposons une interprétation nouvelle de la place qu'assigne Aristote à la technique dans le domaine du savoir, et notamment de son autonomie vis-à-vis des sciences théoriques. Cette interprétation se fonde sur l'analyse par Aristote de la délibération pratique (*βούλευσις*) comme d'une forme d'*investigation* (*ζήτησις*), qui s'oppose à la science (*ἐπιστήμη*) comme savoir constitué et achevé dans sa présentation déductive. La technique apparaît alors comme le savoir positif qui permet de *guider* la délibération, et qui a donc trait aux *méthodes*.

Nous étudions successivement les deux perspectives qu'offre Aristote sur la technique à travers les deux textes mentionnés, en cherchant à les concilier ; *Mét.* A.1 nous permet d'établir que l'universalité propre à la technique est celle des méthodes, prises comme *généralisations de délibérations* (section 6.1), et *Éth. Nic.* VI que leur mode de vérité est celui de la nécessité hypothétique (section 6.2). Cette interprétation étant dégagée, nous montrons qu'Aristote pointe dans l'*Organon* la possibilité de *méthodes systématiques* pour résoudre des problèmes, et notamment au livre I des *Premiers analytiques* où il expose la technique du *Pont aux ânes* (ainsi que la tradition l'a dénommée). Celle-ci nous semble étroitement liée à l'idée de système scientifique, telle qu'elle est dégagée au livre II des *Seconds analytiques* (section 6.3).

Dans notre réflexion finale, nous nous demandons pourquoi Aristote semble avoir réservé la modalité systématique de la connaissance aux savoirs scientifiques, à l'exclusion des savoirs techniques, qui pourtant accèdent bien aussi à l'universel.

6. (Allen 2015, p. 68). Ce point est noté également par Johansen (2017, p. 121).

Nous conjecturons la possibilité de penser des *systèmes techniques* dans un cadre aristotélicien (section 6.4).

6.1 Comment la technique atteint-elle l'universel ?

Cette section est destinée à interpréter la position qu'Aristote assigne à la la technique dans l'échelle du savoir exposée en *Mét.* A.1, entre expérience et science. Nous commençons par revoir les interprétations classiques et récentes qui tendent à y reconnaître une position essentiellement kantienne – à savoir que, si l'art imite la nature, alors le savoir technique n'est qu'une forme dégradée du savoir scientifique, qui se contenterait de le « prendre à l'envers », c'est-à-dire de partir de l'effet pour remonter à la cause (§ 96). Nous examinons ensuite une autre interprétation récente, qui tend à l'assimiler au contraire à l'expérience (§ 97). Afin d'éviter ces deux réductions, nous revenons à l'idée que la technique, tout comme la prudence, a d'abord affaire à la prise de décision *hic et nunc* et qu'il peut donc y avoir des raisonnements pratiques parfaitement rigoureux qui n'enchaînent que des termes particuliers, et ne sont donc nullement une voie d'accès à l'universel (§ 98). La technique, dans cette perspective, doit être vue comme l'universalisation réussie de certaines délibérations, dans des domaines et des types de situations qui se prêtent à une telle opération (§ 99).

§ 96. La possibilité d'une interprétation réductionniste

Le texte principal qui permet la réduction de la technique à une forme subalterne de la science est l'introduction de la *Métaphysique*, qui la décrit comme un degré intermédiaire de la connaissance, s'élevant au-dessus de l'expérience et préfigurant la science. Elle est même *déjà* science, si on la considère relativement à l'expérience : « En général, un signe du savoir ou de non-savoir est la capacité à enseigner, et c'est pourquoi on considère que la technique, plus que l'expérience, est science »[7]. Cette qualification est reprise dans de nombreux autres textes, par exemple : « ...les techniques, et les sciences productives... »[8] et « ...nulle science, qu'elle soit pratique, productive ou théorique... »[9].

7. *Mét.* A.1, 981b8.
8. *Mét.* Θ.2, 1046b3.
9. *Mét.* E.2, 1026b5.

Aristote a un usage du terme *ἐπιστήμη* « qui respire », tantôt restreint à son sens le plus strict, tantôt très large, comme dans cette dernière citation[10]. Il ne s'agit pas d'un emploi lâche, au contraire : comme la connaissance est une affaire de progrès ascendant vers les principes, elle se donne par degrés, et Aristote emploie le terme *ἐπιστήμη* lorsqu'il veut représenter un gradient positif, c'est-à-dire la connaissance s'affermissant.

Dans la *Métaphysique*, c'est plus spécifiquement les sciences *théoriques* qu'il qualifie de supérieures aux sciences productives (982a1), et cela parce qu'elles s'occupent de remonter à l'universel et aux principes des choses[11]. Parmi les sciences théoriques, Aristote cite successivement les mathématiques (982a24), l'astronomie (982b15) – qui est une partie des mathématiques – et enfin la théologie (983a5). À strictement parler, ce sont les seules sciences qui ont pour objet les êtres nécessaires et éternels, et qui peuvent donc produire un discours absolument certain. Mais dans sa classification des sciences théoriques en *Mét.* E.1, Aristote justifie longuement la place de la physique aux côtés des mathématiques et de la théologie. Les êtres naturels en effet, contrairement aux êtres artificiels, portent en eux-mêmes le principe de leur propre mouvement, et l'étude des causes du développement naturel des choses est donc une entreprise théorique.

C'est l'étude des mêmes mécanismes et processus qui occupe également les techniciens qui souhaitent les répliquer ou les modifier à leur avantage. Physiciens et techniciens ont donc, d'une certaine manière, le même domaine d'investigation : « Selon qu'on fabrique une chose, ainsi se produit-elle par nature, et selon que la nature produit une chose, ainsi la fabrique-t-on, à moins d'empêchements » dit *Phys.* II.8, 199a9. Mais leur point de vue est différent : alors que les techniciens étudient les causes en rapport avec le problème à résoudre, les savants étudient les causes d'un point de vue universel, et cherchent ainsi à remonter aux premières causes[12]. Un exemple favori d'Aristote est la médecine, dont il a bien perçu la collaboration nécessaire avec la science :

> Mais c'est aussi l'office du physicien, en ce qui concerne la santé et la maladie, de voir quels sont leurs principes premiers, car ni santé, ni maladie ne peuvent exister dans les êtres privés de vie. Aussi peut-on dire de la plupart des physiciens, et, parmi les médecins, de ceux qui procèdent dans leur technique d'une manière plus

10. Sur ce sujet, voir (Crubellier et Pellegrin 2002, p. 37-40) (Moss 2014, p. 203 et p. 207, n. 33).
11. *Mét.* A.2, 982a25-b3.
12. *Mét.* *α*.1, 994b20-30 ; voir aussi *APo.* I.13 et I.24.

> philosophique, que les premiers aboutissent dans leurs études aux vérités d'ordre médical, et que les derniers, dans les questions de médecine, prennent pour point de départ les principes venant de la science de la nature (*Du sens et des sensibles*, 436a17-436b1).

Aussi Le Blond ([1936] 1970, p. 328) peut-il commenter :

> Il n'y a pas en effet deux façons d'accomplir une production, l'une selon la nature et l'autre selon l'art, mais les deux activités comportent la mise en jeu des mêmes éléments, le parcours des mêmes étapes [...] Ce parallélisme s'explique d'ailleurs par le fait que l'art « imite la nature »[13], et qu'il continue ce qu'elle a commencé[14].

La pratique, lorsqu'elle a accès au savoir scientifique, peut donc raisonner avec la même rigueur, en procédant par des « syllogismes » qui remontent, à l'envers, la chaîne des causes établie par la physique. Le terme *syllogisme* est utilisé à propos des raisonnements technico-pratiques en *Éth. Nic.* VI.13, 1144a31 et dans d'autres textes comme *De anima* III.11, 434a10, ou le *De Motu Animalium*, et le texte d'*Éth. Nic.* VI.13 justifie l'expression spécifique de *syllogisme pratique*. L'idée générale en est bien exprimée par le texte suivant :

> L'homme bien portant résulte [en effet] du raisonnement suivant : puisque la santé est telle-chose, il est nécessaire, pour que la santé soit, que telle-chose arrive, par exemple l'homogénéité, et pour cela, la chaleur. Et le médecin continue de raisonner ainsi jusqu'à ce qu'il arrive à ce qu'il peut produire lui-même. Ensuite le mouvement à partir de là, c'est-à-dire jusqu'à la santé, est appelé production[15].

On voit comment tous ces éléments ont pu conduire une partie du commentaire aristotélicien à minimiser la différence entre science et technique. Si la technique est déjà connaissance des causes, si elle porte sur les choses naturelles tout autant que la physique, si elle raisonne également par syllogismes, en quoi peut-elle donc différer de la science théorique, sinon par une simple différence de degré, ou par son orientation vers la production ? Rodier (1926, p. 194), paraphrasant ce passage, peut donc écrire :

> Savoir que la définition de la santé consiste dans telle proportion ou harmonie des éléments corporels ; que cette proportion suppose la présence dans l'organisme de telle quantité de chaud et de froid [etc.] est le fait de la science.
>
> *Il est facile d'apercevoir que cette science peut servir de fondement à la pratique. Pour réaliser un but nous n'aurons qu'à suivre, dans l'ordre inverse, la série des*

13. *Phys.* II.2, 194a21.
14. Voir à ce sujet l'étude de Le Blond ([1936] 1970, pp. 326-346) sur les *schèmes de l'industrie*, qui montre comment ils informent la conception aristotélicienne des quatre causes.
15. *Mét.* Z.7 1032b1.

> *conditions déterminées par la raison discursive.* […] Cette aptitude à découvrir les moyens dont les rapports nécessaires nous serviront à produire une fin contingente, est précisément la prudence. Ou plutôt, la prudence est cette aptitude appliquée à la réalisation de la vertu. *(Nous soulignons)*

Rodier n'est pas entièrement exact dans son analyse, puisque la médecine est pour Aristote une technique, et non une forme de prudence, d'où sa reprise (« ... ou plutôt... ») en fin de citation. Mais *a fortiori*, son assertion que le raisonnement pratique ne fait qu'inverser l'ordre de la preuve scientifique résonne singulièrement avec la thèse kantienne de la technique et de l'habileté comme *corollaires* du savoir théorique.

Dans une interprétation récente, Jozef Müller, qui vise contre Rodier à réaffirmer l'autonomie de la prudence par rapport au savoir scientifique, reprend pourtant son argument en le circonscrivant cette fois à la seule technique. Le point clé pour Müller est que seule la technique, qui cherche à établir l'universalité de ses méthodes, parvient à une telle nécessité hypothétique dans ses raisonnements :

> Dans sa forme idéalisée, la pensée productive précède par nécessité hypothétique : si A doit être fabriqué, alors, étant donné ce qu'est A, B doit être fait, sans référence aucune aux désirs, préférences, croyances ou valeurs [du producteur]. Sans surprise, Aristote associe de manière répétée le raisonnement par nécessité hypothétique à la production, [...et ainsi] elle ressemble plus à la *theoria* qu'à la *praxis*[16].

L'intérêt pour Müller de cette interprétation est qu'elle permet d'isoler l'élément propre de la prudence (*φρόνησις*), c'est-à-dire sa capacité à déterminer la bonne fin sur laquelle fixer le désir[17]. Car contrairement à la pensée productive, à qui l'objet est *donné*, et donc indifférent – Aristote répète souvent que le médecin est capable de la guérison comme de l'empoisonnement – la pensée pratique est préoccupée de déterminer l'objet du désir, et est donc particulière à chacun et à la situation donnée. Par exemple, une personne pourrait réfléchir ainsi : *je veux rendre ma vie plus confortable, que faire ? – je vais agrandir ma maison*, et l'architecte pourrait l'aider à confirmer cette résolution, en lui fournissant des plans ou un devis, mais cette pensée productive est logiquement consécutive et subordonnée à celle qui fixe le désir et la fin :

> Le raisonnement pratique porte sur ce qu'il faut faire vis-à-vis de ses besoins ou désirs. Lorsqu'il est conclu, il est suivi par une action (faite par décision) qui inclut

16. (Müller 2018, p. 159).
17. Contre l'objection possible de *Éth. Nic.* III.3 (« On ne délibère pas des fins, mais des moyens »), voir, outre l'argumentation propre de Müller, Nussbaum ([1986] 2001, p. 297).

> la pensée productive […] Si la décision est le principe de l'action, alors la pensée productive peut être (et est en fait) une partie de l'action[18].

Müller fait donc de toute la pensée productive une espèce de l'habileté (δεινότης), un terme introduit par Aristote vers la fin du livre VI de l'*Éthique à Nicomaque*. Il traduit de manière significative ce terme par *expertise*, plus général que *cleverness*, habituellement choisi par les traductions anglaises. C'est cette « expertise » qui, pour Müller, est une modalité de la pensée théorique, et qu'il faut contraster avec la délibération pratique. Müller, il est vrai, n'affirme pas cette thèse explicitement, mais il la suggère. Outre le passage cité, il emploie une seconde fois le terme « théorie » dans le contexte de l'opposition de la pensée productive à la pensée pratique[19]. Aussi affirme-t-il qu'Aristote n'associerait la nécessité hypothétique qu'au raisonnement productif[20], et nullement à la délibération pratique, qui n'établit selon lui aucun lien nécessaire entre la fin désirée et l'objectif sur lequel elle la fixe[21].

Cette interprétation n'est pas recevable, car elle introduit une scission au sein de la délibération pratique qui n'existe pas dans le texte aristotélicien. Si on peut admettre, à la suite de l'interprétation initialement proposée par Wiggins (1975) d'*Éth. Nic.* III.3[22], que la délibération fait plus que seulement déterminer les moyens, et qu'elle sert également à fixer le désir sur un objet déterminé, il n'est tout de même nulle part question chez Aristote de rejeter la considération des moyens hors de la délibération de la personne prudente, comme le rappelle avec force Moss (2011). Une telle personne est *aussi* habile[23], et donc maître dans l'art d'agencer les moyens à son avantage. Par ailleurs, *Éth. Nic.* III.3 compare la délibération à un raisonnement géométrique :

> Celui qui délibère semble investiguer et analyser le sujet en question à la manière d'un diagramme. S'il est clair que toute investigation n'est pas délibération – telles les mathématiques – cependant toute délibération est une investigation.
> *ὁ γὰρ βουλευόμενος ἔοικε ζητεῖν καὶ ἀναλύειν τὸν εἰρημένον τρόπον ὥσπερ διάγραμμα* (*φαίνεται δ' ἡ μὲν ζήτησις οὐ πᾶσα εἶναι βούλευσις, οἷον αἱ μαθηματικαί, ἡ δὲ βούλευσις πᾶσα ζήτησις*)[24].

18. (Müller 2018, p. 159).
19. (ibid., p. 162).
20. (ibid., p. 152, n. 9).
21. (ibid., p. 159).
22. Voir plus bas § 100.
23. *Éth. Nic.* VI.12, 1144a29.
24. *Éth. Nic.* III.3, 1112b20.

Cette analogie entre la délibération et les raisonnements des mathématiques suggère que la nécessité hypothétique vaut en général de toute délibération ; elle est d'ailleurs notée à de multiples reprises par Aristote comme un fait remarquable, et notamment en *Phys.* II.9, 200a15 à propos des raisonnements productifs, ce qui montre qu'il n'y a nul lieu, à cet égard, de les distinguer des délibérations en général [25].

L'interprétation de Müller bute également sur l'autre texte majeur du corpus aristotélicien concernant la division entre théorie et pratique, à savoir le développement de l'*Éth. Nic.* VI concernant les facultés et vertus de l'âme rationnelle. Dans ce texte, Aristote insiste de manière répétée sur l'opposition *conjointe* des pensées pratiques et productives à la pensée théorique. Elles relèvent toutes deux d'un même type de réflexion, la délibération (*βούλευσις*) qui est le fait de la partie « calculatrice » (*λογιστικόν*) de l'âme :

> Posons donc qu'il y a deux parties raisonnables de l'âme, l'une par laquelle nous considérons (*θεωροῦμεν*) ces choses dont les principes ne peuvent être autrement, et l'autre celles qui le peuvent, puisque lorsque les choses diffèrent génériquement, des parties génériquement différentes de l'âme leur sont naturellement adaptées. En effet c'est sur la base d'une similitude et d'une appropriation que leur advient la connaissance.
>
> Appelons donc l'une « épistémique » et l'autre « calculatrice ». Car délibérer est la même chose que calculer[26] (*λεγέσθω δὲ τούτων τὸ μὲν ἐπιστημονικὸν τὸ δὲ λογιστικόν· τὸ γὰρ βουλεύεσθαι καὶ λογίζεσθαι ταὐτόν*)[27].

Comme nous l'avons déjà vu[28], une telle caractérisation des objets d'étude de la raison « épistémique » n'exclut pas la physique ; cela est confirmé plus loin dans le livre VI, où Aristote réaffirme que les êtres naturels ne sont pas l'objet de la technique, pour la même raison qu'en *Mét.* E.1[29]. Cependant cette caractérisation met plutôt en avant les sciences théorétiques pures, comme les mathématiques ou l'astronomie. Aristote souhaite ainsi insister sur une différence radicale entre théo-

25. Voir également *De Motu Animalium* 7, 701a6 ; *Éth. Eud.* II.11, 1127b28. En *Mét.* Z.7 l'exemple de raisonnement pratique est très détaillé mais ne mentionne pas explicitement les raisonnements mathématiques.
26. Nous nous autoriserons de cette équivalence fournie par Aristote lui-même pour traduire *λογιστικόν* tantôt par raison calculatrice et tantôt par raison délibérative.
27. *Éth. Nic.* VI.1, 1139a5-12.
28. Voir plus haut, § 96.
29. « La technique ne concerne ni les choses qui sont nécessairement, ni les choses qui sont naturellement, car celles-ci ont en elles leur principe »[30].

rie et pratique, ce qui est confirmé par la précision qu'il s'agit d'une « différence générique » (*τὰ τῷ γένει ἕτερα*), la différence la plus haute dans la logique aristotélicienne, radicalité qui se répercute sur la distinction établie entre les deux formes de l'âme rationnelle.

§ 97. L'interprétation de Lorenz et Morison

Une telle radicalité a amené une interprétation récente (Lorenz et Morison 2019) à avancer la thèse qu'Aristote présenterait ici deux épistémologies indépendantes, l'une rationaliste, fondant la connaissance sur la démonstration déductive à partir des premiers principes, et l'autre inspirée de l'empiricisme médical de son temps, la fondant sur l'observation et l'inférence prédictive. Il n'est pas difficile de voir qu'une telle interprétation est également, à sa manière, réductionniste : car elle ne fonde plus la distinction entre connaissance théorique et connaissance pratique sur la différence qu'il y aurait à connaître les choses en tant que telles et les choses en tant qu'elles sont « effectuables », mais plutôt, au sein même du domaine de la connaissance théorique, sur la différence entre deux méthodes antithétiques pour établir le savoir. Au-delà des paradoxes que cela suscite, par exemple qu'une même personne pourrait connaître deux fois le même fait[31], cette lecture fait violence aux nombreux textes aristotéliciens, dont certains ont déjà été cités, qui promeuvent une vision unifiée de la connaissance : les *Analytiques* décrivent la science comme tout entière fondée sur l'expérience, et ne parvenant aux premiers principes qu'après un long cheminement, du « plus connaissable pour nous » au « plus connaissable en soi ».

Lorenz et Morison fondent en partie leur argumentation sur le fait qu'Aristote utilise également le terme de « raison opinante » (*λογός δοξαστικόν*) pour désigner la raison calculatrice[32]. Aussi identifient-ils la connaissance pratique tout entière avec la connaissance empirique en général. On ne voit alors plus très bien la différence entre technique et expérience ; l'exemple qu'ils donnent d'une connaissance technique ressemble en fait étrangement à ce qu'Aristote appelle expérience en *Mét.* A.1. Ainsi, comme exemple de cause accessible à la technique, ils donnent la raison inductive suivante, tirée de Galien : « parce qu'il a été observé que, si quelque chose est relâché dans telles conditions, alors un spasme survient(ibid., p. 45) ». Cependant, un tel énoncé ne capture pas la cause selon Aristote, mais

31. Voir la section 4, qui tente de répondre aux objections portant principalement sur ce fait.
32. Par exemple, *Éth. Nic.* VI.5, 1140b26.

décrit une expérience générale non fondée.

Plus généralement Aristote définit la technique justement par le fait de s'élever au-dessus de l'expérience par la perception des causes. Aristote emploie le terme δοξαστικόν pour désigner la partie rationnelle pratique de l'âme dans des contextes qui ont pour objet la prudence et l'habileté ; celles-ci opèrent en effet au niveau de l'expérience. Mais le terme λογιστικόν est le terme générique qu'il utilise pour désigner la faculté pratique en général, et celle-ci inclut la technique.

Le terme clé est donc λογιστικόν. λογισμός signifie « compte » ou « calcul » dit le dictionnaire Bailly, mais s'entend en deux sens, qui résonnent encore dans les termes français : le premier, associé à l'idée de nombre, donne le calcul mathématique, mais le second, sans idée de nombre, donne lieu à des expressions comme *tenir compte de..., faire quelque chose avec la pensée que si..., autant qu'il était possible selon les calculs de la raison humaine...*, etc. C'est évidemment en ce second sens qu'Aristote affirme que délibérer et calculer sont synonymes. Il s'agit donc du calcul qui a lieu dans les affaires courantes, celui qui évalue la valeur des choses ou la chance d'une opportunité, *qui établit des plans d'action*. Loin d'appliquer mécaniquement les règles d'un calcul mathématique, il est plutôt celui qui manœuvre de manière flexible entre les faits particuliers pour établir un résultat chaque fois singulier. Cette forme de réflexion s'oppose donc fortement à l'attitude épistémique, dont on verra, un peu plus loin, qu'elle s'occupe seulement de démonstration déductive. Le « syllogisme pratique » semble donc recéler une différence plus radicale avec le syllogisme simple que celle d'une simple inversion du raisonnement.

Il y a ainsi une tension entre d'une part la gradation « verticale » de *Mét.* A.1, qui présente la technique comme un premier degré de la science, et insiste donc sur leur continuité, et d'autre part la distinction *générique* de l'âme rationnelle effectuée dans l'*Éth. Nic.* VI, qu'on pourrait qualifier « d'horizontale », et qui insiste sur leur discontinuité. C'est cette tension que nous devons comprendre plus avant à présent, en tenant compte de la perspective et de l'intention de chaque texte.

§ 98. L'à-propos des syllogismes pratiques

Comment donc comprendre que, selon *Mét.* A.1 la technique *est déjà science*, comme nous l'avons déjà dit[33], tout en procédant d'une rationalité génériquement

33. Voir plus haut, § 96.

distincte ?

La cohérence entre les deux points de vue pourrait être assurée si on arrive à montrer que

(i) le mouvement qui permet de s'élever de l'expérience à la technique, et celui qui permet de s'élever à la science (que ce soit à partir de la technique, ou de l'expérience directement) sont génériquement distincts ;

(ii) le premier mouvement peut néanmoins inspirer le second et lui être utile.

Cependant la proposition (i) semble rien moins qu'évidente, puisqu'Aristote semble dire que la technique atteint l'universel, comme la science, par la saisie de la cause : « [Les techniciens] « voient la cause (*τὴν αἰτίαν ἴσασιν*) », et [les personnes d'expérience] non[34] ». Pour résoudre cette difficulté, il nous faut ici revenir à la question du syllogisme pratique, puisque saisir la cause, c'est établir un syllogisme entre les termes qui décrivent le phénomène[35] ; peut-être qu'en saisissant ce qui distingue syllogisme pratique et syllogisme scientifique, nous distinguerons mieux en quoi la cause saisie par la technique diffère de celle de la science.

Or on doit immédiatement remarquer, que contrairement au syllogisme scientifique, le syllogisme pratique n'est pas nécessairement une voie d'accès à l'universel. Même en laissant de côté les syllogismes en forme d'application de règle (*rule-case syllogisms* dans le commentaire anglo-saxon)[36], les syllogismes qui sont issus de la délibération pratique, et notamment ceux de la personne habile ou prudente, traitent de situations *ad hoc*, qui ne se laissent pas généraliser aisément. Une délibération typique procède en effet selon la méthode de l'analyse, décrite par Aristote en *Éth. Nic.* III.3 :

> Nous ne délibérons pas des fins, mais de ce qui contribue à la fin [*ἀλλὰ περὶ τῶν πρὸς τὰ τέλη*] […] On pose la fin et on regarde comment et par quels moyens [*διὰ τίνων*] on peut l'atteindre, et, si plusieurs moyens paraissent en mesure de l'atteindre, on examine quel est le plus facile et le plus beau. Mais s'il n'y en a qu'un seul pour arriver à cette fin, on cherche comment il permet d'y arriver et aussi par quel moyen ce moyen lui-même peut être atteint, jusqu'à pouvoir parvenir à la première cause, qui est la dernière dans l'ordre de la découverte[37].

34. *Mét.* A.1, 985a28.
35. *APo.* II.4, 91a15.
36. Par exemple dans le *De Motu, 7, 701a 13* : Tout homme doit se promener, or je suis un homme, donc je vais me promener.
37. *Éth. Nic.* III.3, 1112b12.

Dans cette heuristique de la fin et des moyens, le raisonnement peut rester confiné à l'enchaînement de termes particuliers – nous développons un exemple plus loin. La cause dont il est question dans le texte n'est pas nécessairement une liaison universelle entre des termes généraux, mais un moyen (*διὰ τί*) dans ce que nous appellerions aujourd'hui un *plan d'action* plutôt qu'un syllogisme.

Ce qu'indique cependant Aristote par l'emploi du terme *syllogisme* pour désigner les plans d'action obtenus par délibération rationnelle, c'est d'abord l'enchaînement nécessaire des conséquences des actions ; comme nous allons le suggérer plus loin, mieux encore que « plan d'action » lui correspondrait notre idée moderne d'une *procédure*. Ce que « syllogisme » suggère également, c'est l'idée d'une convertibilité entre une procédure et une démonstration *achevées*, mais non de l'équivalence de leur formation. Ce point a été vu par Lorenz et Morison :

> La comparaison entre les options, qui était une partie du processus de délibération, n'est pas capturée par le syllogisme pratique. Celui-ci présente les résultats de la réflexion pratique (les choix faits et les raisons pour lesquelles ils ont été faits) mais il ne présente pas la vision complète de comment ces résultats furent obtenus[38].

Si donc les plans d'actions sont le fruit des délibérations pratiques, qu'est-ce qui fait que certains sont susceptibles d'universalisation pour devenir *techniques*, tandis que d'autres, notamment ceux qui résultent de la réflexion éthique ou politique, ne le peuvent pas ?

L'interprétation de Müller, malgré les limites que nous lui avons trouvé[39], nous met sur la piste : ce qui permet à un raisonnement productif d'être universalisé, dit-il, c'est que les circonstances particulières et subjectives de l'agent *n'y comptent pour rien*, qu'un problème productif est le problème de *quiconque* :

> Les résultats [de la pensée productive] – les recettes pour produire ou fabriquer quelque chose – sont universels. Ils ne sont liés à aucun individu en particulier et sont valides et vrais pour quiconque désire produire les produits visés et est capable de suivre les instructions pour ce faire[40].

Cet argument, pris tel quel, n'est pas suffisant : tout raisonnement à visée productive n'est pas nécessairement universalisable parce qu'il est désubjectivisé. Le négociant qui détermine la meilleure route pour son voyage, tenant compte de mille circonstances commerciales et logistiques particulières, tient un raisonnement qui n'a aucune dimension personnelle (il ferait de même s'il était salarié d'un

38. (Lorenz et Morison 2019, p. 447).
39. Voir plus haut, § 96.
40. (Müller 2018, p. 152).

armateur), mais qui ne se laisse pas universaliser pour autant. C'est donc plus généralement encore qu'il faut prendre l'argument de Müller : ce ne sont pas seulement les raisonnements engageant le désir personnel qui ne sont pas susceptibles d'être universalisés, mais plus généralement tous ceux qui dépendent de manière déterminante de circonstances qu'il n'est pas possible de subsumer sous une règle générale – comme certes la circonstance de mon propre désir, qui est malléable et capable de se reconfigurer justement en fonction des situations[41], mais également comme toute autre circonstance qui résiste à sa réduction à un cas générique simple. Par exemple, dans le cas du négociant, il y aura le fait que la saison est propice pour la navigation, mais aussi que des pirates ont été récemment signalés vers Égine, qu'un navire au capitaine expérimenté pourrait partir bientôt, etc. Non seulement ces circonstances sont interdépendantes et leur évaluation *conjointe* délicate à établir, mais il faut également savoir que ce sont justement celles-ci qu'il est pertinent de considérer, parmi la myriade d'autres faits disponibles. Cet *à-propos* de la délibération est justement le fait de l'expérience, dit Aristote, et c'est ce qui la distingue essentiellement des raisonnements mathématiques, malgré l'analogie formelle qui les rapproche. La prouesse démonstrative des jeunes gens, nous dit Aristote, ne les rend pas sages pour autant :

> […] Les jeunes gens arrivent à maîtriser la géométrie ou les mathématiques et à devenir sages dans les matières de ce genre, alors que, semble-t-il, on n'arrive pas à être prudent à cet âge. Le motif en est que la prudence comporte aussi une connaissance des choses particulières, lesquelles deviennent familières avec l'expérience[42].

Ce n'est pas le raisonnement ou le type de nécessité qui distinguent les mathématiques de la délibération, mais leur *espace de problème* qui est bien plus vaste dans le second cas, et qui nécessite qu'on sache s'y repérer, bien plus, qu'on sache d'abord comment le circonscrire de manière appropriée dans chaque situation.

Dans le passage du *Pont aux ânes* dans les *Premiers analytiques*[43] qui établit la méthode de construction de tous les syllogismes possibles entre deux termes, Aristote déclare que cette méthode « est la même aussi bien en philosophie qu'en toute sorte de technique (*τέχνη*) et toute discipline »[44]. Mais il ajoute aussitôt que

41. « Par exemple Randy a envie de cacahuètes, mais n'en trouve pas à la maison. Comme il est trop tard (pense Randy) pour sortir en acheter, il décide de prendre un thé à la place » (ibid., p. 159).
42. *Éth. Nic.* VI.9, 1142a12.
43. Sur le *Pont aux ânes*, voir plus loin notre développement § 104.
44. *APr.* I.30, 46a3.

les listes élémentaires de termes sur lesquelles cette méthode opère doivent être fournis par l'expérience ; et c'est justement ces listes qu'il est difficile d'établir une fois pour toutes dans la plupart des délibérations pratiques.

Ce n'est donc pas la forme syllogistique ou le type de nécessité qui distinguent les délibérations pratiques des raisonnements scientifiques, mais leur particularité. Allen écrit justement :

> Le résultat des délibérations [de la personne prudente] sera une décision justifiée par un syllogisme. En comparaison d'un syllogisme scientifique, ce syllogisme apparaîtra comme distinctement *ad hoc*. Les considérations qui emporteront la décision seront, à coup sûr, générales […] mais la raison pour laquelle cette considération doit l'emporter plutôt que d'autres qui, comme généralités sont tout autant applicables, est spécifique à l'occasion, et ne pourra pas être formulée et ré-appliquée comme un principe général elle-même[45].

On voit à cela qu'une technique peut émerger lorsque, partant de multiples délibérations similaires, elle parvient à décrire leur espace de problème de manière uniforme, tout en simplifiant et en réduisant radicalement le nombre de termes nécessaires à cette description, et surtout en garantissant leur stabilité. D'après Nussbaum ([1986] 2001, p. 94-96), la technique, dans le monde Grec du v^e^ siècle, se définit exactement comme antithèse et antidote à la *fortune* (*τύχη*), comme « l'application délibérée de l'intelligence humaine à un domaine du monde, visant à y gagner quelque contrôle sur la fortune ». Il s'agit de rendre les choses prédictibles, contrôlables, précises, et donc assignables à un concept universel fixe. L'élimination de la variabilité des circonstances est l'objet même, et donc la première condition, de la technique. La technique vise plus que des méthodes, elle vise des *procédures* – si nous acceptons de traduire par ce concept moderne le « syllogisme pratique » d'Aristote. On peut définir provisoirement une procédure comme une suite de prescriptions opératoires enchaînées rigoureusement, produisant l'effet attendu de manière déterminée, une fois les circonstances de la situation initiale correctement décrites, ce qui nous semble correspondre à la nécessité hypothétique qui caractérise le syllogisme pratique. Les procédures rendent la méthode accessible aux personnes qui ne possèdent pas le savoir-faire, mais seulement la maîtrise des gestes les plus courants et les plus simples de la communauté humaine à laquelle elle s'adresse.

Aussi, afin qu'il y ait technique, il ne suffit pas, comme l'avance Müller, que

45. (Allen 2015, p. 67).

sa fin puisse être déterminée indépendamment de toute circonstance particulière liée à l'agent ; doivent également être susceptibles d'une description générale les moyens et toute caractéristique de la situation qui sera mobilisée par la procédure technique : un moyen dont la qualité pourrait varier, ou dont la quantité pourrait être limitée, un geste délicat ou incertain, une circonstance qui modifierait le cours de l'action, tous ces éléments ruineraient la procédure du technicien, à moins d'être spécifiés comme tels, et intégrés comme *contraintes* à l'objectif lui-même. La visée du technicien n'est jamais le produit seulement mais un objectif qui comprend aussi bien le produit que les moyens à disposition et les circonstances de l'action. Sa réflexion consiste à réduire, autant que possible, la vaste richesse des vérités particulières fournies par l'expérience à un petit nombre de principes, à *spécifier* les fins, les moyens et les circonstances de manière à les rendre univoques. Dans ce champ clos où chaque terme a sa signification précise, la délibération peut commencer à ressembler vraiment à une investigation mathématique ; c'est pour cela que la technique seulement, et non la prudence ni l'habileté en général, est la porte d'accès à l'universel et le commencement de la science.

§ 99. La technique et la prudence

Ce qui point donc à l'horizon de l'idée aristotélicienne de la technique, ce sont nos concepts modernes de *spécification* et de *standardisation*, tels que les comprend l'ingénieur moderne. C'est la spécification qui réalise l'universel dans le champ pratique et qui correspond à une nécessité hypothétique orientée vers l'action : *Si le résultat recherché est tel, les moyens disponibles tels, et les circonstances telles (*spécification*) alors procédez ainsi (*méthode*)* dit la technique. Ce qui fait qu'Aristote reste en deçà de ces concepts modernes, c'est, au-delà sans doute des facteurs culturels et technologiques de son temps, sa sensibilité propre à la contingence de certains métiers comme la navigation et la stratégie, dont l'art lui semble justement être de s'adapter à des conditions favorables[46]. Plus profondément encore, pour Aristote la technique ne peut jamais s'affranchir de l'expérience, car même une fois décrites les situations pertinentes à l'emploi d'une méthode, encore faut-il savoir reconnaître ces situations dans le cas concret. C'est pour cela qu'Aristote écrit qu'un technicien, qui aurait appris son art de manière seulement

46. Sensibilité sans doute supérieure à celle de Platon, cf. (Angier 2010, p. 38). Voir plus généralement le développement à ce sujet de Dunne ([1993] 1997, p. 254-258), mitigeant l'analyse par M. Nussbaum de l'antithèse *techne-tuche*.

théorique, échouerait dans sa pratique et serait moins efficace qu'une simple personne d'expérience[47].

Ainsi notre thèse est la suivante : *les techniques, essentiellement, ont pour domaine propre de connaissance les objets résultant de l'universalisation de certaines délibérations, c'est-à-dire les méthodes et les procédures*. Les procédures sont les manières générales de produire un résultat général, qu'il s'agisse de fabriquer de la poterie, de jouer de la flûte, de naviguer, etc. Les méthodes guident la délibération particulière de façon à l'abréger, voire à la rendre inutile. *A minima*, elles lui fournissent une liste de moyens à essayer, peut-être plus ou moins ordonnée, comme le font les *Topiques* pour la discussion dialectique. Au mieux, elles lui donnent des procédures complètes, qu'il n'y a plus qu'à appliquer à la situation présente – ce qui peut avoir sa propre complexité – pour obtenir le résultat recherché, c'est-à-dire le plan approprié. « L'art ne délibère pas ! », affirme *Phys.* II.8, 199b28 de manière apparemment contradictoire avec tous les développements d'*Éth. Nic.* VI : l'art ne délibère pas en effet, lorsque la méthode est si bien arrêtée et assimilée qu'il n'y a plus besoin de réflexion pour agir, comme pour l'orthographe et la gymnastique[48].

Cette thèse permet enfin d'expliquer les propositions (i) et (ii)[49] que l'universalité qu'accomplit la technique est génériquement différente de celle qui a lieu dans la science, et qu'elle peut néanmoins être un tremplin pour la science. Une procédure n'est pas une démonstration, avons-nous déjà dit, même si elles sont convertibles l'une dans l'autre, par inversion de l'ordre des causes[50]. Pour mieux le voir encore, prenons pour fil directeur un des exemples favoris d'Aristote, la construction d'une maison. L'homme d'expérience qui aura participé à la construction de plusieurs maisons aura sans doute appris à jauger le poids d'une charpente et à évaluer en conséquence, par exemple, la section nécessaire des colonnes et des murs destinés à la soutenir. Il pourra sans doute également prolonger ses estimations sur diverses variations autour du plan type qu'il connaît. L'architecte, par contre – par exemple le maître d'œuvre des cathédrales, connaissant les principes de géométrie constructive, pourra déterminer méthodiquement quelles sont les structures stables à employer, et quelle est la taille adéquate des pierres, *quelle que soit la configuration du bâtiment à réaliser*. Procédant par méthodes, il peut combiner et faire varier librement les moyens universels qu'elle lui fournit, au sein

47. *Mét.* A.1, 981a20.
48. *Éth. Nic.* III.3, 1112b2-6.
49. Voir plus haut, § 98.
50. Voir plus haut, § 96.

de l'espace de problème qu'elle délimite. Il n'en connaît cependant ni l'essence ni la raison (la gravité, la portance, la tension des matériaux...), qui sont l'objet des recherches du physicien. À proprement parler, le technicien ne connaît nullement des causes, mais seulement des moyens (διὰ τί), qui sont, si on peut dire, *comme des candidats au statut de cause*, soumis à la validation du physicien. L'exemple de l'architecture des cathédrales montre combien une technique admirable peut être éloignée de la découverte des causes véritables ; on dira par exemple que le cercle ou le carré assurent la stabilité des constructions, assertions qui ne sont pas fausses en un sens, mais qui ne touchent pas aux causes profondes. Le cercle et le carré sont plutôt des outils de la méthode, et sont l'occasion d'une théorie provisoire, d'un « bricolage » de la pensée, selon l'expression célèbre de Claude Levi-Strauss.

La dialectique entre outil (ὄργανον) et cause (αἰτία) articule donc le double statut de la technique. Outil et cause sont les moyen-termes recherchés, respectivement, dans l'enquête délibérative et dans l'enquête théorique, et ils sont donc entièrement distincts l'un de l'autre[51]. En même temps – ce qui justifie la proposition (ii) – ils sont convertibles l'un dans l'autre : toute cause naturelle peut être mobilisée comme outil par le technicien, et tout outil est à tout le moins une hypothèse de causalité pour le scientifique. La technique, comme enquête des méthodes universelles, procède donc d'une démarche radicalement différente de celle de la science, tout en collaborant étroitement avec elle – en lui fournissant des indices quant aux causes des choses, et en mobilisant ses résultats en retour.

6.2 L'être-vrai de la technique

On peut à présent revenir au livre VI de l'*Éthique à Nicomaque* afin de comprendre l'originalité radicale de la technique relativement aux autres modes de connaissance, qui fonde la distinction entre théorie et pratique qu'Aristote nous a léguée. Elle se distingue de la prudence en ce qu'elle n'a jamais affaire qu'à des fins déterminées par ailleurs – à un *cahier des charges* dirions-nous aujourd'hui, ou encore une *spécification*[52] (§ 100) mais également de la science en ce qu'elle n'atteint jamais les vraies causes, sinon par accident, mais seulement les *moyens* de l'action (§ 101). Nous concluons cette section par une vue d'ensemble sur le statut de la technique chez Aristote, connaissance à proprement parler *moyenne*,

51. *Éth. Nic.* III.3, 1112b29 affirme que l'outil est bien l'objet de l'investigation délibérative.
52. Voir plus haut, section 3.3.

c'est-à-dire située au carrefour de la connaissance – entre théorie et pratique d'une part, entre science et expérience de l'autre. On a du mal à ne pas penser qu'Aristote la jugeait en cela *médiocre*, tandis qu'elle nous apparaît aujourd'hui comme *centrale* à tout projet de connaissance (§ 102).

§ 100. L'être-vrai de la technique

L'objet du livre VI est de caractériser la réflexion humaine au sein de l'action éthique ; elle doit donc être d'abord distinguée à la fois de la réflexion pure (qui pour Aristote est théorique) et de la réflexion qui détermine l'action sans caractère éthique, celle qui est purement utilitaire. Les chapitres 1 à 5 réalisent l'essentiel de ce programme, par la définition successive de la science (*ἐπιστήμη*), de la technique (*τέχνη*) et de la prudence (*φρόνησις*). Si cette caractérisation passe en partie par la description de leurs modes de raisonnement spécifiques, (délibération *vs* démonstration), ou de leur « psychologie » (relation au désir et à la sensation) c'est la détermination du *mode de vérité* propre à chacune, pensons-nous, qui est au centre des considérations d'Aristote.

D'abord, au chapitre 1[53], lorsqu'Aristote distingue deux parties dans l'âme rationnelle, c'est en référence implicite à l'idée traditionnelle dans la philosophie grecque que dans la vérité, la pensée devient égale à son objet : à objets différents, parties de l'âme différentes[54]. Or ce sont bien les modalités possibles de ce rapport que vise Aristote quand il insiste, tout au long du livre VI, sur la distinction des objets des connaissances théorique, technique et pratique. On a vu déjà en effet[55] que le médecin et le biologiste étudiaient tous deux la même classe de phénomènes, mais sous un angle différent. On peut aussi aisément imaginer, à leur côté, un homme politique s'intéressant à la guérison d'un malade important, par exemple un général pouvant sauver la cité. L'objet (le général malade) peut donc être *matériellement* le même dans les trois cas, mais la réflexion ne s'y rapporte pas de la même manière : ce qui signifie, pour Aristote, que cette même matière est l'occasion de trois objets *formellement* différents : les causes biologiques de la maladie, ses traitements possibles, et ses conséquences pour la cité. Il reste cependant à expliquer en quoi ces trois rapports au même objet matériel sont formellement

53. Texte cité plus haut § 96.
54. Sur la généalogie de cette idée et son traitement par Aristote, on peut consulter la note de J. Tricot sur ce passage de l'*Éth. Nic.* VI.1, 1139a10, p. 275 de son édition, n. 5.
55. Voir plus haut, § 96.

différents – et pas simplement, par exemple, une simple différence de point de vue.

Le chapitre 2 donne une première caractérisation de cette différence par le fait que, dans la sphère pratique, la vérité et l'erreur sont corrélés au désir correct, alors qu'elles valent absolument dans la sphère théorique. Cette corrélation est en fait une co-détermination : la raison vraie et le désir correct se supposent et se renforcent mutuellement[56]. Mais de ce fait, cette détermination est insuffisante, car la notion de désir correct est elle-même recherchée. Par ailleurs, elle ne permet pas encore de distinguer la réflexion pratique de la réflexion productive, qui ne vise que des biens relatifs.

Le chapitre 3 propose donc un nouveau départ, en partant explicitement cette fois de la notion de vérité, telle qu'Aristote l'a déterminée dans le traité *de l'Interprétation*, c'est-à-dire comme propriété spécifique du jugement affirmatif ou négatif[57]. Il s'agit de déterminer plus précisément les dispositions *propres* qu'ont les deux parties rationnelles de l'âme à saisir la vérité, ce qu'Aristote appelle leurs « vertus ». Une telle démarche permet de distinguer explicitement l'être-vrai qui a lieu dans la réflexion productive de celui qui a lieu dans la réflexion pratique. Par ailleurs, en fixant sa recherche sur les *dispositions propres* de chaque mode de réflexion, Aristote signifie qu'il est préoccupé de trouver l'essence de leur être-vrai, et non l'être-vrai par accident, comme l'opinion qui est écartée immédiatement. La prudence (*φρόνησις*) et la technique (*τέχνη*) sont ainsi les deux « vertus » de la raison calculatrice, et du côté « épistémique », Aristote ajoute à la science (*ἐπιστήμη*) proprement dite l'intellect (*νοῦς*) et la sagesse (*σοφία*). La sagesse sera présentée comme la conjonction de la science et de l'intellect, lui-même étant défini comme la saisie intellectuelle immédiate des principes. Seules donc la science, la prudence et la technique sont des dispositions *discursives* à saisir le vrai, accompagnées de *logos*, c'est-à-dire d'une raison ayant une double valeur normative et explicative[58].

Ces trois modalités de saisie de la vérité sont donc définies successivement, avec des formulations qu'Aristote cherche à rendre les plus homogènes possibles, dans les trois sections consacrées à la science (*ἐπιστήμη*) (VI.3), la technique (*τέχνη*) (VI.4) et la prudence (*φρόνησις*) (VI.5). Voici les passages centraux :

> La science est une disposition démonstrative, il faut voir tout ce que nous avons dit là-dessus dans les *Analytiques* ; on connaît scientifiquement en effet lorsqu'on ac-

56. *Éth. Nic.* VI.2, 1139a24.
57. *Éth. Nic.* VI.3, 1139b14.
58. Voir à ce sujet la mise au point de Moss (2014) sur le sens de *λογός* dans l'*Éthique à Nicomaque*.

> quiert une forme de croyance et de familiarité avec les principes ; et si celle-ci n'est pas supérieure à celle de la conclusion, c'est par accident qu'on a la science.
> *ἡ μὲν ἄρα ἐπιστήμη ἐστὶν ἕξις ἀποδεικτική, καὶ ὅσα ἄλλα προσδιοριζόμεθα ἐν τοῖς ἀναλυτικοῖς. ὅταν γάρ πως πιστεύῃ καὶ γνώριμοι αὐτῷ ὦσιν αἱ ἀρχαί, ἐπίσταται. εἰ γὰρ μὴ μᾶλλον τοῦ συμπεράσματος, κατὰ συμβεβηκὸς ἕξει τὴν ἐπιστήμην.* (1139b.31)
> Toute technique a pour objet la génération, exercer une technique c'est donc considérer comment générer une de ces choses qui peuvent être ou n'être pas[...] La technique est donc, comme nous l'avons dit, une disposition productive, accompagnée de raison vraie, de choses contingentes. Le manque de technique, au contraire, est une disposition productive basée sur une raison fausse.
> *ἡ τέχνη* [*ἐστὶν*] *ἕξις μετὰ λόγου ἀληθοῦς ποιητική. ἔστι δὲ τέχνη πᾶσα περὶ γένεσιν καὶ τὸ τεχνάζειν καὶ θεωρεῖν ὅπως ἂν γένηταί τι τῶν ἐνδεχομένων καὶ εἶναι καὶ μὴ εἶναι.* [...] *ἡ μὲν οὖν τέχνη, ὥσπερ εἴρηται, ἕξις τις μετὰ λόγου ἀληθοῦς ποιητική ἐστιν, ἡ δ' ἀτεχνία τοὐναντίον μετὰ λόγου ψευδοῦς ποιητικὴ ἕξις, περὶ τὸ ἐνδεχόμενον ἄλλως ἔχειν.* (1140a.9)
> La prudence est une disposition pragmatique vraie, accompagnée de raison, concernant les biens et les maux de l'homme.
> [*ἡ φρόνησις*] *εἶναι ἕξιν ἀληθῆ μετὰ λόγου πρακτικὴν περὶ τὰ ἀνθρώπῳ ἀγαθὰ καὶ κακά.* (1140b5)

Ces textes doivent être mis en rapport avec la théorie du jugement affirmatif et négatif, qu'Aristote a rappelée en liminaire de ces développements. La doctrine aristotélicienne de la vérité et de l'erreur[59] est construite sur cette théorie : si juger affirmativement, c'est lier deux concepts, et juger négativement les délier, alors selon que cette liaison est dans les choses ou non, on pourra être dans la vérité ou l'erreur.

Cette perspective éclaire la définition de la prudence : il s'agit de juger de l'action considérée en rapport « aux biens et aux maux » humains ; affirmer qu'elle est conforme aux biens, ou au contraire qu'elle conduit à des maux. Qu'un tel jugement puisse être qualifié de vrai ou de faux se fonde sur le fait que les biens et les maux humains sont dans l'absolu déterminés par l'essence même de ce que c'est que d'être une personne humaine. Mais ils sont également dépendants de la situation actuelle dans laquelle se situe le sujet, des options qui lui sont ouvertes, des moyens à mettre en œuvre qui peuvent avoir des conséquences néfastes, des événements qui pourraient survenir dans le futur, etc. Aussi en général aucune certitude n'est possible concernant ces jugements. C'est pour cela, à notre avis, que

59. *Mét.* Θ.10, 1051b1 ;b35, *de l'Interprétation*, 4-6.

l'adjectif « vrai » qualifie ici la disposition de l'âme et non la raison qui l'accompagne, contrairement à la définition de la technique ; Aristote indique par là que si le raisonnement de la prudence est faillible, sa garantie de vérité procède d'abord de l'attitude générale que l'homme vertueux adopte dans ses délibérations.

Ce mode de vérité contraste singulièrement avec celui de la technique, qui est pourtant le fait de la même raison calculatrice. La différence majeure réside dans le critère qui permet de juger du plan d'action : dans le domaine technique, le critère n'est rien d'autre que le résultat recherché, ou encore « le cahier des charges » tel qu'il a été posé au départ de la réflexion. Johansen dit exactement cela :

> Le vrai *logos* impliqué dans la technique représente donc un raisonnement correct concernant les moyens qui réalisent un certain produit[60].

Si le technicien juge correctement de l'adéquation des moyens envisagés au cahier des charges, alors il est dans la vérité, et sinon, il est dans l'erreur, dans l'*ἀτεχνία* dit Aristote, terme qui ne signifie pas nécessairement que son projet échouera, mais qu'il ne réussira pas grâce au *logos*. On verra par exemple plus loin[61] que les aveux extorqués par la torture sont atechniques relativement à la rhétorique, qui est l'art de convaincre ; cela ne veut pas dire, malheureusement, que la torture n'est pas efficace. La perspective d'Aristote n'est pas de mettre en avant l'efficacité de la technique mais d'expliciter ses modes d'être vrai et faux.

Le mode de vérité de la délibération technique diffère radicalement de celui que vise la prudence. Même si celle-ci s'occupe également de délibérer des moyens, elle réalise en amont de cela un travail important, qui est de *spécifier* à quoi ressemble une bonne solution, quels sont justement les critères qu'elle devrait satisfaire. Elle réalise ce travail de spécification en prenant en compte les finalités immédiates et de long-terme de la personne ou de la communauté dont elle a charge, réalisant ainsi nécessairement des choix pour ce faire. Aussi Wiggins écrit-il :

> Le problème standard dans une délibération non-technique est assez différent. Dans un cas non-technique, j'aurai typiquement une description extrêmement vague de ce que je veux – une bonne vie, une profession satisfaisante, des vacances intéressantes, une soirée amusante – et le problème n'est pas de savoir ce qui sera causalement efficace pour réaliser cela, mais de voir ce que serait une spécification adéquate et réaliste pratiquement qui satisferait ce besoin. La délibération est encore *zetesis*, une recherche (*search*), mais elle n'est pas d'abord une recherche de moyens. Elle est une recherche de la *meilleure spécification*. […] Quand elle est trouvée, alors la

60. (Johansen 2017, p. 101).
61. Voir plus haut, § 103.

> délibération concernant la fin et les moyens peut commencer, mais les difficultés qui surviennent dans cette délibération peuvent me renvoyer plusieurs fois au problème d'une meilleure ou plus pratique spécification[62].

On retrouve ici le terme *spécification* que nous avons utilisé plus haut pour caractériser la technique. Alors que le technicien ne commence à travailler que lorsqu'il a reçu une spécification du résultat qu'il doit viser et des ressources dont il dispose – par exemple le charpentier reçoit la spécification du gouvernail par le pilote[63], l'homme prudent est celui qui *écrit* la spécification, et qui l'écrit *librement*, au regard de ce qu'il pense être le bien pour lui-même ou pour la communauté qu'il sert, et non sous la contrainte d'un autre objectif imposé par quiconque – cette précision pour le distinguer d'un expert qui ne ferait que conseiller les désirs d'un maître, ce qui serait encore une activité technique. Ce faisant, comme le remarque Wiggins, s'instaure un dialogue entre les deux phases de la délibération, qui peut conduire à de multiples itérations : car l'homme prudent observe également les effets collatéraux des moyens que sa réflexion suggère, et peut y voir des conséquences rédhibitoires, ou au contraire des opportunités inaperçues jusqu'ici. La réflexion pratique est donc sans cesse occupée d'un libre « va-et-vient » des fins vers leurs moyens, et des moyens vers leurs conséquences, l'effort tout entier de la prudence étant d'*intégrer* tous ces éléments dans une perspective holistique qui maximiserait le bien, en tenant compte des chances et des incertitudes. C'est pour cela que, d'une certaine manière, dit Aristote, la prudence est réalisée de manière excellente chez les hommes politiques tels que Périclès, qui possèdent une capacité intégrative de jugement inégalée[64].

On voit ici pourquoi l'interprétation de Müller déjà discutée[65] est réductrice : en établissant une scission forte entre les deux étapes de la délibération, et en identifiant la délibération prudente strictement à la première, elle méconnaît les cas où il est difficile, voire impossible, de les distinguer, car la complexité de la situation requiert un réajustement constant des finalités au vu des options considérées. Tout l'objet de la technique est d'obtenir une telle scission entre le donneur d'ordre et le producteur, en délimitant un espace de production maîtrisé qui impose également le langage dans lequel doivent s'exprimer les besoins. Mais une telle technicisation n'est pas toujours possible dans le domaine de la réflexion productive. Aussi l'in-

62. (Wiggins 1975, p. 38).
63. *Phys.* II.2, 194a36.
64. *Éth. Nic.* VI.5, 1140b7.
65. Voir plus haut, § 96.

terprétation de Müller réduit-elle à ce cas idéal la réalité beaucoup plus fluide de la délibération. Là ne saurait donc résider la distinction entre la prudence et la technique. Elle tient, bien plus essentiellement, à ce que la première se situe dans un espace de délibération essentiellement *ouvert*, où elle doit juger, du mieux qu'elle peut, *ce qui sera pour le mieux*, tenant compte de toutes les contingences pertinentes qu'elle peut déceler dans la situation présente, et de l'ensemble de ce que son expérience et sa délibération lui ont appris, tandis que la seconde se tient dans un espace *clos*, dont les termes ont des valeurs précises et immuables, et qu'elle doit seulement tenter d'arranger entre eux de la meilleure manière possible. La vérité de la première vise l'horizon indéfiniment ouvert du bien humain, tandis que celle de la seconde se borne à la satisfaction d'une commande.

§ 101. Savoir technique et savoir scientifique

On peut à présent contraster ces deux modes de vérité avec la vérité par excellence, la vérité scientifique. Ici, les éléments donnés par Aristote sont parcellaires ; il se contente d'y énoncer qu'une condition du savoir démonstratif est que les principes en soient mieux connus que la conclusion, et renvoie aux *Analytiques*. Cette assertion conclut en effet le long développement d'*APo*. I.2 qui a pour objet de définir ce qu'est la connaissance scientifique. Son point de départ, c'est la thèse célèbre du Stagirite, que connaître vraiment (scientifiquement), c'est connaître la cause d'une chose et pourquoi elle ne peut être autrement qu'elle n'est. C'est pour cela que le syllogisme est essentiel à la science, car il lie un jugement à un principe tiers qui le justifie ; et si ce principe est établi par ailleurs, ou déjà connu, alors le syllogisme peut être qualifié de démonstration. La vérité scientifique vise à approfondir les causes des choses, c'est-à-dire à entrer plus profondément dans le « ce qu'elle est », sa structure interne ou ses principes de comportement, et à passer de ce qui est plus connaissable pour nous (i.e. par la perception) à ce qui est plus connaissable « par nature »[66]. La vérité scientifique est donc tout entière dans cette adéquation de la pensée avec la cause et la structure intime de la chose ; et c'est pour cela que, pour être accomplie, elle nécessite aussi bien la science que l'intellection (*νοῦς*) des principes, c'est-à-dire la sagesse théorique qui les conjugue toutes deux.

Si donc le scientifique et le technicien connaissent tous deux la cause d'une chose – par exemple que la chaleur est la cause de la digestion – ils n'ont absolu-

66. *APo*. I.2, 71b33-72a5.

ment pas la même perspective sur elle. Le premier cherchera à remonter plus avant dans la compréhension de cette loi, tandis que le technicien ne s'y intéressera que comme un état de fait qu'il peut mobiliser dans la résolution de son problème. Les résultats de la science ne sont rien d'autre pour le technicien que des « moyens » supplémentaires pour la détermination de ses procédures. Nous suivons le point de vue de Johansen qui écrit :

> Nous pouvons nous attendre à ce que le *logos* de la technique comprenne des éléments d'information théorique, mais seulement dans la mesure où ils sont instructifs pour la production. Une bonne manière de penser à ce *logos* est de penser à une recette. Si vous voulez faire de la crème glacée de manière experte, ou enseigner cette compétence à quelqu'un, vous devez avoir connaissance des processus d'émulsion, de stabilisation et de congélation. […]. Un technicien aristotélicien pourrait savoir que les protéines sont des stabilisateurs, et en cela il diffèrera du glacier simplement expérimenté qui sait seulement que le blanc d'œuf a permis par le passé de rendre le mélange plus onctueux. Mais le technicien n'a pas besoin de connaître la chimie précise à l'œuvre, et cela parce qu'il est seulement intéressé à faire un produit précis, de la crème glacée[67].

Dans ce texte Johansen exprime bien pourquoi la technique ne s'élève pas à la considération des causes premières : c'est parce qu'elle s'arrête à l'intérêt de son besoin ; cela ne l'empêche pas de poser des questions à la science, ou de rechercher plus avant les causes qui lui semblent pouvoir être utiles à son entreprise ; mais là n'est pas sa perspective originelle, qui est seulement d'établir des plans efficaces pour atteindre le résultat visé.

La vérité de la technique est la vérité du raisonnement, avons-nous dit plus haut, qui agence correctement les moyens aux fins, et qui ne demande à être jugé que sur la pertinence de cet agencement ; que les moyens viennent à manquer dans telle situation n'altère en aucun cas la vérité du raisonnement, pas plus que le théorème de Pythagore ne devient faux face à un triangle équilatéral. Elle est essentiellement, comme le répète souvent Aristote (voir par exemple *Phys.* II.9) *nécessité conditionnelle*, du type « Si vous voulez ceci, alors faites cela ». Elle vise la réalité telle qu'elle pourrait être, à l'opposé de la visée de la science, qui essaie de comprendre les choses *telles qu'elles sont*, ce qui explique le choix d'Aristote dans le livre VI de l'*Éthique à Nicomaque* de dramatiser l'opposition entre la science « au sens strict »[68], comme occupée uniquement des êtres nécessaires, et entre la

67. (Johansen 2017, p. 125).
68. *Éth. Nic.* VI.3, 1139b19.

technique et la prudence comme occupées du contingent.

Il faut enfin remarquer sur ce point que c'est la prudence, à certains égards, qui dans la philosophie d'Aristote doit être rapprochée de la science, plutôt que la technique. Si la vérité de la prudence est en effet, comme pour la technique, l'adéquation de l'action envisagée avec la fin, celle-ci n'est pas ici une fin déterminée, mais la fin générale de la personne humaine, son bien essentiel, « objectif » dirions-nous aujourd'hui. C'est pour cette raison, à notre avis, que la conclusion du livre VI de l'*Éthique à Nicomaque* distingue finalement, parmi les cinq vertus intellectuelles introduites initialement, deux vertus suprêmes seulement, la sagesse théorique et la prudence[69]. Car la première vise le réel tel qu'il est en soi, et la seconde vise le bien réel de l'homme. Cela est signifié à cet endroit du texte par les expressions fortes, à connotation platonicienne, qui décrivent la prudence comme « œil de l'âme » (*ὄμματι τῆς ψυχῆς*), à qui seul « le bien le plus noble » apparaît (*τὸ τέλος καὶ τὸ ἄριστον*).

Des cinq manières d'être-vrai énumérées initialement, seule donc la technique a vraiment disparu, puisque science et intellection sont conjointes dans la sagesse. Cette exclusion n'est-elle due qu'à un préjugé grec contre l'artisanat, dont on sait qu'Aristote n'est pas exempt[70] ? Ou est-elle plutôt liée à sa modalité spécifique d'être-vrai, qu'il jugerait inférieure aux deux autres ?

Tel nous semble en effet le cas. Adéquation de la pensée à la chose, adéquation des moyens envisagés aux objectifs fixés, adéquation de l'action envisagée au bien de l'homme : si on résume ainsi les modalités respectives de l'être-vrai de la science, de la technique et de la prudence, on voit d'emblée que la seconde ne se situe pas au même niveau que les deux autres, qui visent l'être ou le bien objectifs. La technique au contraire, seulement préoccupée d'agencer des processus causaux déterminés en vue de fins déterminées elles aussi, se meut dans des espaces de problème clos, où elle n'a finalement affaire qu'à elle-même. Sa vérité est fondamentalement la vérité de la nécessité conditionnelle « si …alors ».

L'exclusion de cette modalité de la vérité est justifiée de manière explicite par Aristote en *Éth. Nic.* VI.12 où il argumente en faveur de la suprématie de la sagesse et de la prudence sur les autres vertus. Il oppose alors la vertu à l'habileté (*δεινότης*), dont on a vu[71] qu'elle devait être considérée comme une modalité ou

69. *Éth. Nic.* VI.11, 1143b15.
70. *Pol.* III.3.
71. Voir plus haut, § 96.

une proche parente de la technique. Toutes deux en effet se distinguent de la prudence en ce qu'elles sont indifférentes aux fins, en ce que le médecin est capable de la guérison comme de l'empoisonnement, et en ce que la personne habile peut être aussi bien prudente que rouée, selon qu'elle possède la vertu ou non. Aussi, conclut Aristote, on peut être habile et néanmoins dans l'erreur, car « le vice pervertit et falsifie (διαψεύδεσθαι) la saisie des principes de l'action » (1144a34). La vérité conditionnelle de l'habileté et de la technique n'est que cela : vraie seulement si la fin qu'elles visent est-elle même bonne, mais de cela elles ne peuvent rien savoir. Indifférentes donc aussi bien aux causes profondes des choses qu'aux fins dernières de la personne humaine, elles ne sont capables que de cette demi-vérité que donne la nécessité conditionnelle, qui n'a aucune valeur en soi, mais seulement d'utilité, à la science qu'elle stimule, et à la prudence qu'elle assiste.

§ 102. Le statut inférieur de la technique

Notre interprétation permet donc d'expliquer la tension formelle de laquelle nous sommes partis, entre schéma vertical et schéma horizontal de la connaissance rationnelle, et d'expliquer l'originalité de la connaissance technique vis-à-vis de la connaissance théorique, tout en la distinguant également fermement de la réflexion pratique, aussi bien que de l'expérience en général.

Il ne fait aucun doute que ces deux schémas étaient compatibles et cohérents dans l'esprit du Stagirite. La réflexion délibérative est généralement la compétence propre de la personne d'expérience, car elle suppose le bon jugement des circonstances. Certaines délibérations, qu'Aristote appelle productives, ont leur objet déterminé par un concept fixé empiriquement, comme « la santé » ou « une selle de cheval », plutôt que par des fins particulières comme : « comment rembourser l'argent à mon créancier ? » – ou par des idées générales du bien comme le « bonheur » ou l' « honnetêté », dont le sens dépendra nécessairement de chaque situation. Lorsqu'une méthode est établie par la délibération productive, qui mobilise des moyens eux aussi décrits par des concepts généraux, et que la connexion entre ces moyens et la fin visée est établie causalement, on accède alors à la connaissance technique. La vérité propre de ce type de connaissance est celle de la nécessité hypothétique, c'est-à-dire, celle de la cohérence interne entre la spécification du problème et la méthode de résolution proposée. Ce mode de vérité est inférieur à la vérité « proprement dite », c'est-à-dire orientée vers le réel, qui a elle-même deux modes : celui de la sagesse théorique qui réside dans l'adéquation de la pen-

sée avec les choses, et celui de la prudence pratique qui est dans la juste perception du bien à réaliser dans la situation présente. En particulier, les moyens que mobilisent les méthodes techniques – ses « outils » – ne doivent être considérés que comme des *indices* de causes, comme des pointeurs vers une théorie possible, et non comme des éléments attestés de cette théorie. Ils peuvent être totalement reconfigurés, voire discrédités par la vérité scientifique, quels que soient leurs succès pratiques.

Il faut noter que cette infériorité de la technique contraste avec la position centrale qu'elle occupe dans chacun des deux schémas, entre science et prudence d'une part, et entre expérience et science théorique d'autre part. Une telle situation aurait pu évoquer l'idée de la technique comme d'un « carrefour » de la connaissance, permettant le commerce entre ses domaines propres : c'est en prenant une forme technique qu'un savoir empirique ou pratique peut être étudié théoriquement, et réciproquement aussi bien. Mais pour une fois chez Aristote, la position médiane n'est pas un signe de prééminence. La technique est non seulement inférieure à la science quant à la saisie des causes, et à la prudence quant à la considération du bien, mais elle est également inférieure à l'expérience quant à l'efficacité, qui est pourtant sa raison d'être. La technique apparaît donc comme une forme mineure de la rationalité, limitée dans sa hauteur de vue, bornée dans l'horizon de son action, myope dans le jugement des circonstances.

C'est à notre avis là que réside la tension essentielle interne au concept de technique chez Aristote, domaine autonome du savoir, et en même temps, forme primitive et mineure du savoir, qui n'a pas la large assise de l'expérience, ni l'élévation de la science. C'est cette tension qui a amené si souvent les commentateurs, et en tous cas la tradition philosophique, à l'assimiler tantôt à l'une, tantôt à l'autre, méconnaissant sa particularité d'être un *savoir des méthodes et des procédures*.

6.3 Méthode et logique

Il est à présent possible de revenir à notre problématique principale. On peut se demander en effet si la *systématisation* de la technique, par laquelle nous avons caractérisé au chapitre précédent l'approche industrielle du travail et la programmation des ordinateurs, est tout simplement compréhensible, voire possible d'un point de vue aristotélicien. La technique, visant à universaliser des délibérations particulières qu'elle abstrait de situations contingentes, peut-elle aller jusqu'à de-

venir *systématique* – si on prend ce terme pour l'instant en son sens intuitif ?

Nous exposons d'abord pourquoi cette possibilité semble peu crédible, en prenant l'exemple d'une technique particulière, la rhétorique, dont il nous reste un traité complet d'Aristote (§ 103). Il y a cependant un lieu où on voit Aristote chercher à mettre en place une démarche systématique d'investigation : l'*Organon*, où il expose des techniques pour trouver les arguments adéquats à la résolution de problèmes donnés, notamment dans les *Topiques* et les *Premiers analytiques* (§ 104). Le *Pont aux ânes*, une méthode pour trouver les moyens termes des syllogismes, nous intéresse particulièrement par son caractère nettement procédural et, avons-nous envie de dire, « systématique », d'autant plus qu'Aristote la déclare pertinente aussi bien « pour la philosophie [que] pour les techniques et toutes les disciplines ». Or l'idée de *systématicité* est d'abord associée à la doctrine aristotélicienne de la science exposée par les *Seconds analytiques*. Nous tentons donc de montrer en quoi ces deux idées sont liées, et pourquoi la résolution systématique de problème que propose le *Pont aux ânes* peut le mieux être envisagée dans le cadre d'un système scientifique (§ 105). Cela nous permet de dégager enfin une notion unifiée de l'investigation, quoiqu'elle se trouve dramatiquement différenciée entre science et technique (§ 106).

C'est sur cette base que nous tenterons, à la section suivante, de comprendre la possibilité de *systèmes techniques* dans une perspective aristotélicienne.

§ 103. Les méthodes de l'investigation

Notre résultat principal est que ce sont les techniques qui sont les dépositaires du savoir pratique, et que leurs objets sont les *méthodes*. Nous avons utilisé jusqu'ici ce dernier terme de manière informelle, et dans son sens moderne. Or *méthode* est un terme important pour Aristote ; d'après le catalogue de ses œuvres transmis par Diogène Laërce, il aurait écrit un traité *des Méthodes* en huit livres, classé aux côtés des œuvres de l'*Organon*, et un autre traité *de la Méthode*, qui suivait trois traités concernant la technique, malheureusement tous perdus. Sans répondre à notre question de manière catégorique, étant données ces lacunes du corpus aristotélicien, on peut néanmoins *raisonner à partir d'Aristote*, de manière à éclairer notre sujet.

C'est donc l'idée de *méthode* qu'il faut creuser pour comprendre ce que peut être le savoir technique. Comme le remarque Lennox (2011, p. 29), μέθοδος est un terme difficile à traduire, car il se colore de nuances diverses en fonction du

contexte. Il apparaît surtout en introduction des traités, lors des considérations réflexives que fait Aristote sur la démarche que doit adopter l'étude face à l'objet en question. Il se rapproche alors de son sens actuel, en ce qu'il recouvre les préceptes qu'Aristote se donne concernant l'investigation. Il peut également prendre le sens plus général de *discipline, enquête, investigation*, comme en *Phys.* I.184a10 : « regardant toute étude (*theoria*) et toute *μέθοδος* », ou les premiers mots de l'*Éth. Nic.* I.1, 1094a1 : « Toute technique et toute *μέθοδος* ». Il apparaît, dans ce deuxième sens, interchangeable avec *σκέπτεσθαι* et surtout *ζήτησις* qu'Aristote utilise très couramment, ainsi qu'*ἱστόρια*, moins fréquent, pour désigner de manière réflexive ses propres recherches, que celles-ci portent sur la philosophie première, la physique, les disciplines éthiques ou logiques. *σκέπτεσθαι* et *ζήτησις* ont leurs nuances propres (*examen* / *investigation* respectivement) mais sont également traduits, selon les contextes, par *investigation* ou *enquête*. Aussi *μέθοδος*, *ἱστόρια*, *ζήτησις* et *σκέπτεσθαι* sont aisément interchangés dans ces textes réflexifs, sans qu'on puisse y déceler de nuance significative. Tous ces termes désignent le savoir en train de se constituer activement, par rapport au savoir déjà constitué que sont les sciences et les techniques.

Investigation (*ζήτησις*) est le terme qu'Aristote utilise pour décrire la recherche de la cause[72] et du moyen-terme[73] comme essence de l'activité scientifique. C'est également le terme qu'il utilise dans le passage déjà cité d'*Éth. Nic.* III.3[74], lorsqu'il déclare que la délibération est une investigation, mais que toute investigation n'est pas une délibération, en donnant les mathématiques comme contre-exemple. Allen (2015) et Charles (2015) ont tous deux relevé ce parallèle, en comparant *Éth. Nic.* III.3 à *APr.* I.27-30 et *APo.* II.1-2 ;8 respectivement. Nous l'avons nous-même déjà évoqué[75], afin d'expliquer que procédures et démonstrations se laissent décrire par la même forme du syllogisme.

Si la délibération est une investigation, elle n'est donc pas un savoir, mais un savoir en train de se réaliser, le savoir de ce qu'il faut faire dans la situation actuelle. La méthode se situe alors à un niveau réflexif qui guide la délibération, tout comme elle peut guider le scientifique qui cherche à expliquer un phénomène. Si le syllogisme pratique représente la procédure achevée, la méthode, elle, est concer-

72. *Mét.* Z.17.
73. *APo.* II.1-2 ;8.
74. Voir plus haut, § 96.
75. Voir plus haut, § 99.

née par l'écriture de la procédure.

On peut se faire une idée de ce qu'est la méthode d'une technique en examinant la *Rhétorique*. Au début de l'ouvrage, Aristote affirme que « la méthode propre à cette technique concerne les moyens de persuasion (*ἡ ἔντεχνος μέθοδος περὶ τὰς πίστεις ἐστίν*)[76] » et précise plus loin :

> Parmi les moyens de persuasion cependant, les uns sont non techniques (*ἄτεχνοί*), et les autres techniques (*ἔντεχνοι*). J'entends par non techniques tous les éléments qui ne sont pas fournis par nos soins, mais qui existaient au préalable, comme les témoins, les aveux extorqués par la torture, les pièces écrites et autres preuves du même genre. Par techniques, j'entends tous les éléments qu'il nous est possible de mettre en place par la méthode et par nos soins (*ἔντεχνα δὲ ὅσα διὰ τῆς μεθόδου καὶ δι' ἡμῶν κατασκευασθῆναι δυνατόν*). Ainsi n'a-t-on qu'à utiliser les premiers, mais a-t-on à découvrir les seconds (*ὥστε δεῖ τούτων τοῖς μὲν χρήσασθαι, τὰ δὲ εὑρεῖν*)[77].

Appartient donc au domaine de la technique ce dont on est responsable (*ce dont on est la cause*, dit Aristote plus loin[78]), et qui est conduit avec méthode. On peut noter la circularité de la définition, qui caractérise la méthode par ses objets (les moyens de persuasion) et les objets par la méthode, ce qui montre la proximité entre les deux notions.

Or que contient le traité d'Aristote ? Certainement pas une compilation de *syllogismes pratiques* qui pourraient prendre la forme, dans le cas de la rhétorique, de *canevas* de discours qu'il suffirait d'adapter aux circonstances – ce que feront nombre de traités de rhétorique par la suite. Il contient par contre une distinction des principales finalités de la rhétorique, l'examen (souvent normatif) des moyens et des figures de discours disponibles pour le rhéteur, l'analyse des circonstances dont il doit tenir compte, comme les passions de son auditoire, le type d'arguments que porte l'adversaire et comment y répondre, etc.

La méthode de la rhétorique ne dicte donc pas l'écriture du discours, mais elle la cadre et la guide ; elle aide le rhéteur à réfléchir, en lui posant des questions : *« es-tu certain de ce que tu cherches à accomplir ? as-tu étudié ton auditoire ? as-tu pensé à telle figure argumentative ? ton style respecte-t-il telles et telles normes ? etc. »* Elle consiste donc à l'aider à décrire son « espace de problème », c'est-à-dire à répertorier l'ensemble des termes qu'il doit considérer pour trouver une solution – ici un discours persuasif.

76. *Rh.* I.1, 1355a4.
77. *Rh.* I.2, 1355b35.
78. *Rh.* III.16, 1416b19.

Ce résultat est conforme à l'idée générale concernant la technique qui se dégage de notre étude d'Aristote. Une méthode pour parvenir à ses fins n'est pas en général absolument déterminante de l'action ni infaillible. Le rhéteur, le navigateur, le menuisier, ne peuvent s'en remettre à des livres de procédures pour réaliser leurs travaux, cependant leurs techniques les guident dans leur réflexion, peuvent leur rappeler certains points d'attention ou mettre en exergue certains cas particuliers. Cette vision aristotélicienne est exactement celle que nous avons dégagée au chapitre précédent concernant les procédures humaines, lorsque nous avons voulu les contraster avec les procédures industrielles et informatiques[79]. Mais alors comment faire place, dans cette perspective, à la possibilité d'une *action systématique* ?

§ 104. Le Pont aux ânes

Qu'il n'y ait pas de schémas procéduraux dans la *Rhétorique* ne prouve pas que ceux-ci seraient également incongrus en toute autre technique. La proximité de la rhétorique avec les affaires humaines – éthique, politique, droit – peut expliquer ce fait ; mais comme l'incertitude propre à chaque activité productive varie, elles pourraient être possibles et légitimes pour d'autres techniques.

Et de fait, on trouve bien de tels schémas dans ce que nous pouvons deviner d'une doctrine aristotélicienne des méthodes, qu'exposaient peut-être les traités perdus. On peut trouver ces indices dans l'*Organon*, qui n'est pas seulement une somme d'exposés théoriques sur la forme de la proposition et du raisonnement, mais qui contient également, au moins pour les *Topiques* et les *Premiers analytiques*, de nombreuses considérations méthodiques.

Les *Topiques* en particulier se présentent comme un traité *technique* portant sur la dialectique, qui a entre autres une fonction d'entraînement et de formation à la dispute oratoire[80]. Ici, encore, la construction d'une méthode est l'objectif recherché :

> Le présent traité se propose de trouver une méthode (*μέθοδος*) qui nous rendra capable de raisonner déductivement, en prenant appui sur les idées admises, etc.[81].

disent les premières lignes du traité. Ici, la méthode prend un tour beaucoup plus « systématique ». Le livre VIII, qui porte sur la manière de conduire l'entretien

79. Voir plus haut section 5.1 et 5.2.
80. *Top*. I.2, 101a25-b4. Voir l'introduction, p. 13 et suivantes, de P. Pellegrin à son édition (2015).
81. *Top*. I.1, 100a18.

dialectique, a un ton nettement procédural. L'essentiel du traité est composé de listes de lieux, dont l'objet est d'alimenter en arguments un dispositif de raisonnement systématique, décrit par De Pater (1965). Brunschwig décrit le lieu comme « une machine à faire des prémisses à partir d'une conclusion donnée[82] ». Cette démarche procédurale est poussée à son terme en *APr.* I.27-30, qui constitue un mini-traité de « pratique » du syllogisme. Crubellier (2014, pp. 31-34) montre le remarquable progrès que cette démarche présente par rapport aux recettes dialectiques des *Topiques*. Aristote y expose une méthode pour trouver le moyen-terme qui lie deux termes entre eux par un syllogisme. Cette méthode, connue au Moyen Âge par le nom de « Pont aux ânes »[83] est représentée, depuis Jean Philopon, par la figure 6.1. Pour trouver les moyens-termes qui permettent de relier le prédicat A au sujet E, six listes de termes sont établies : B, C, D sont les listes de termes respectivement conséquents, antécédents et incompatibles avec A, et F, G, H ceux qui le sont avec E, dans le même ordre. Si un même terme se retrouve dans deux listes reliées entre elles dans le schéma, alors le syllogisme de la figure indiquée sur la ligne peut être établi. Par ailleurs, il y a un ordre de l'exploration (CF et BG, puis HB et DF, puis GD et GC) qui permet de trouver d'abord le syllogisme le plus informatif.

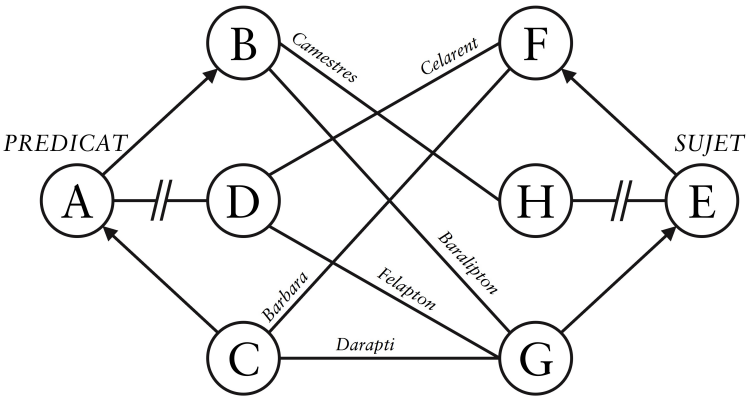


Figure 6.1 – Schéma général du Pont aux ânes

Nous donnons deux exemples du fonctionnement du *Pont aux Ânes*, avec des

82. Cité par P. Pellegrin dans l'introduction de son édition des *Topiques*, p. 26.
83. Voir (West et H. D. Thompson 1891) pour une histoire de ce terme et les exposés de (Hamblin 1976) et de Crubellier dans son édition des *Premiers analytiques*, p. 31-34, 286-294.

listes de deux termes chacune, représentées directement dans le graphique en regard de l'élément correspondant du diagramme.

Le premier exemple (6.2) permet de construire le syllogisme en *Barbara* suivant :

Tout ce qui consomme de l'oxygène est vivant.
Tout homme consomme de l'oxygène.
Donc, *tout homme est vivant.*

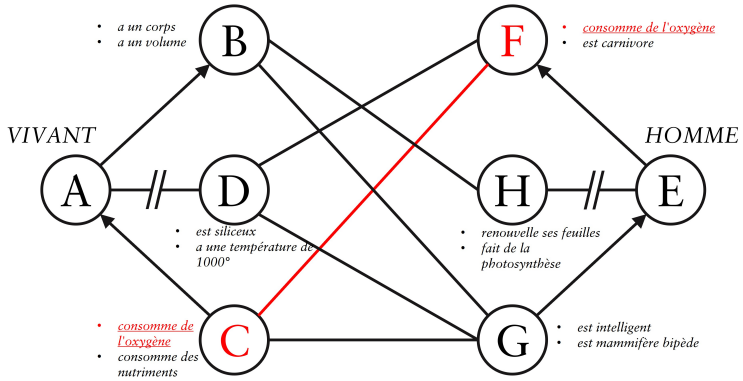


FIGURE 6.2 – Pont aux ânes - exemple 1

Le second exemple (6.3), lointainement inspiré de la *Logique de Port Royal*, permet de construire le syllogisme en *Camestres* suivant :

Tout homme sauvé confesse ses péchés.
Nul esprit fort ne confesse ses péchés.
Donc, *nul esprit fort n'est sauvé.*

On peut remarquer que ce syllogisme est préférable au syllogisme en *Felapton* également possible :

Nul homme sauvé n'est libertin.
Quelque esprit fort est libertin.
Donc, *quelque esprit fort n'est pas sauvé.*

L'ordre d'inspection des listes permet de trouver d'abord les meilleurs syllogismes.

Cette méthode présente un caractère proprement procédural et systématique, qui permet à Aristote de conclure son exposé de manière satisfaite :

> On voit donc, à partir de ce qui vient d'être dit, non seulement qu'il est possible de constituer toutes les déductions par cette méthode, mais aussi qu'il est impossible d'en constituer par une autre voie. En effet, il a été établi que toute déduction

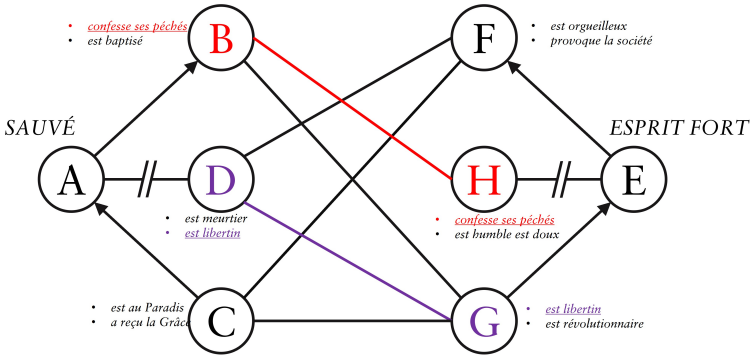


FIGURE 6.3 – Pont aux ânes - exemple 2

est constituée par l'une des figures dont on a parlé auparavant, or il n'est pas possible de les construire autrement qu'au moyen des termes impliqués et de ceux qui impliquent chaque terme[84].

Un peu plus loin, il met en avant un autre bénéfice de cette démarche : elle permet de donner toutes les relations possibles entre deux termes *une fois pour toutes*, sans qu'il soit nécessaire de recommencer la recherche à chaque fois, selon qu'on cherche à affirmer ou à nier une thèse, à établir des faits universels ou des faits particuliers[85].

On voit ici poindre l'idée de ce que nous appellerions aujourd'hui une *démarche systématique d'investigation* : Le *Pont aux ânes*, donc, du moment qu'on lui fournit des listes de termes décrivant exhaustivement un domaine du savoir, peut procéder comme une « machine théorique », écrit Crubellier, dont il est possible de « décrire précisément la structure et les opérations » et qu'on pourrait imaginer de réaliser aujourd'hui « sous la forme d'un moteur de recherche très simple associé à des bases de données[86] ». Surtout cette « machine » ne laisse rien de côté et permet d'identifier toutes les possibilités de résolution du problème en cours.

La méthode est « théorique », selon Crubellier, dans le sens qu'elle dépend de listes de termes issus de l'expérience qui, dès que le domaine enquêté est suffisamment large, deviennent impossibles en pratique à constituer. Il écrit :

En dépit du ton très assuré [d'Aristote], on ne voit pas comment [cette] condition

84. *APr.* I.29, 45b35.
85. *APr.* I.30, 46a10-17.
86. Éd. des *Premiers analytiques* par M. Crubellier, p. 32-33.

> […] pourrait être réalisée en pratique, ni comment on sera certain qu'elle est effectivement réalisée[87].

Avant d'en venir à cette question, cette dépendance envers des listes de termes indique à quel type d'investigation s'adresse le *Pont aux ânes* : il s'agit de la *résolution de problème*. Le terme *πρόβλημα* est en effet utilisé de manière systématique dans ce passage, et il a un sens précis pour Aristote :

> Au sens technique développé dans les *Topiques* un problème se présente sous la forme d'une interrogation double : « est-ce que X est *p* ou *non-p* ? » et sa solution consiste à trouver un syllogisme qui prouve la position que l'on défend[88].

Par sa forme extrêmement restrictive, le problème aristotélicien n'est qu'un lointain antécédent de notre concept actuel. En particulier, il ne semble pas y avoir ici de référence à l'autre sens du terme développé par les géomètres grecs à la même époque pour désigner les questions de construction de figure[89]. On y trouve cependant l'idée cruciale qu'un problème détermine un *espace de problème*, dans lequel doit être trouvée sa solution (ici les listes de termes). Au sein de cet espace, l'enquête se présente comme une *résolution de problème* qui peut être procédurale.

D'où pourraient donc venir ces listes de termes que requiert le Pont aux ânes ? D'abord, il faut remarquer que l'exigence d'exhaustivité est relative au contexte du problème à résoudre. Le début du chapitre *APr.* I.30, consacré à des réflexions épistémologiques sur cette méthode[90], en mentionne deux : celui de la recherche de la vérité, *i.e.* le contexte scientifique, et celui de l'argumentation dialectique, *i.e.* le contexte de l'opinion probable – une opposition classique dans la réflexion d'Aristote. Or le début de ce texte, qui semble se placer dans ce second contexte, dit simplement qu'il faut « en posséder le plus possible ». Dettel et Crubellier renvoient ici à *Top.* I.14 qui décrit de manière concrète ce qui semble avoir été une pratique scolaire courante, et qui deviendra celle du florilège au Moyen Âge :

> Il faut encore recueillir des prémisses dans les livres, et dresser des tableaux pour chaque catégorie de sujets, avec des têtes de rubriques séparées, par exemple *« le bien »* ou *« l'animal »*, « bien » devant être pris dans toute son ampleur, en com-

87. (Crubellier 2014, p. 33, n. 2). Crubellier exprime son doute en disant que le *Pont aux ânes* « ne donne pas une méthode procédurale, c'est-à-dire un ensemble de règles qu'il suffirait de suivre attentivement pour être assuré de réussir » (p. 34), mais il vise là l'impossibilité de réduire en procédures l'expérience elle-même, et nullement la méthode centrale du *Pont aux ânes* qui s'en nourrit et qui, elle, est bien procédurale.

88. Pellegrin, notes à son édition (2005) des *Seconds Analytiques*, p. 416.

89. Voir plus loin, section 9.2.

90. Il s'agit du titre proposé par Michel Crubellier dans sa traduction, p. 136.

> mençant par l'essence. On indiquera en marge, à chaque fois, le nom des tenants de ces opinions [...][91]

Le *Pont aux ânes*, en tant qu'il se situe dans la prolongation des méthodes de l'argumentation dialectique décrites dans les *Topiques*, utilise le même matériau qu'elles. Il est plus efficace et plus exhaustif *relativement à elles*, et non dans l'absolu.

C'est cependant dans le contexte scientifique des *Analytiques* qu'Aristote expose le *Pont aux ânes*, et c'est à l'investigation scientifique qu'il semble la trouver la plus adaptée. À la fin du chapitre *APr.* I.30, il donne l'exemple de l'astronomie, la science par excellence, pour montrer le bien-fondé de sa méthode :

> Nous avons exposé [...] de quelle façon il faut pourchasser les termes des syllogismes, pour éviter ainsi que notre attention ne se porte sur la totalité de ce qui est dit dans les termes du problème ; [...] Ce qu'il faut c'est considérer un plus petit nombre de termes, bien déterminés. [...] Il appartient à l'expérience de fournir les principes afférents à chaque sujet. Je veux dire que, par exemple, c'est l'expérience astronomique qui fournit les principes de la science astronomique, car ce n'est qu'une fois les phénomènes célestes convenablement appréhendés, que les démonstrations de l'astronomie ont été découvertes. Il en est de même pour n'importe quel autre art ou science. Par suite, [...] il nous appartient de dégager promptement les démonstrations. Si, en effet, aucun des véritables attributs appartenant aux choses n'a été omis de notre étude, nous serons capable, en tout ce qui admet une preuve, de la découvrir et de démontrer [...][92]

Pour Aristote, c'est la science elle-même qui doit être la meilleure pourvoyeuse de listes de termes pour le *Pont aux ânes*. Cette méthode sera donc systématique, *dans la mesure où la science le sera elle-même*. C'est au sens d'une telle perspective que nous consacrons notre prochain développement.

§ 105. La science et la systématicité

C'est donc les caractéristiques de la science qu'il faut examiner pour comprendre ce qui rend possible une démarche systématique de résolution de problème. Bien qu'Aristote n'emploie pas le terme *système* pour désigner la forme du savoir scientifique, c'est bien à lui que se réfère Christian Wolff quand il en fixe le

91. *Top.* I.14, 105b13.
92. *APr.* I.30, 46a10-26, traduction Tricot.

sens dans l'acception qui nous est familière aujourd'hui[93] :

> Il ressort de son *Organon* que, parmi les anciens philosophes, c'est Aristote qui avait un esprit systématique. Si on éclaire sa lecture par les notions nécessaires à sa compréhension, il s'avère que l'*Organon* est rédigé de telle sorte que chaque vérité est démontrée par les autres et, qu'en même temps, l'idée d'un esprit systématique se dégage des doctrines présentées elles-mêmes[94].

Cette vision a été contestée par les commentateurs plus récents. Jonathan Barnes, un commentateur contemporain, fait état de cette opinion :

> Le système d'Aristote – son « image du monde » – a suscité pendant des siècles l'admiration et l'éloge. Certains chercheurs ont toutefois contesté cette vision d'Aristote. Ils ont nié qu'il ait été un bâtisseur de systèmes. Se méfiant eux-mêmes des prétentions grandioses de la philosophie systématique, ils estiment que les vertus d'Aristote sont ailleurs. Pour eux, la philosophie d'Aristote est essentiellement « aporétique » : elle consiste à poser des énigmes particulières ou *aporiai*, et à y développer des solutions particulières. La pensée d'Aristote est provisoire, flexible, changeante. […] Cette interprétation anti-systématique de la pensée d'Aristote est aujourd'hui largement acceptée[95].

Barnes a sans doute à l'esprit, entre autres, l'interprétation de Pierre Aubenque[96], inspirée de Heidegger[97]. Contre cette perspective, Barnes maintient la vue de Wolff, notamment quant à sa conception de la science :

> La théorie de la science exposée dans les *Seconds analytiques* ne peut être rejetée comme un archaïsme sans intérêt […] Il y a tant d'allusions et d'indications de systématisation dans les traités que la solution des *aporiai* ne peut pas être considérée comme la finalité des recherches scientifiques et philosophiques d'Aristote ; et – un point qui mérite d'être souligné – même les discussions fragmentaires de problèmes individuels se voient conférer une unité intellectuelle par le cadre conceptuel commun dans lequel ils sont examinés et traités. La systématisation n'est pas réalisée dans les traités, mais elle est un idéal, toujours présent à l'arrière-plan[98].

93. Plus précisément, la référence à Aristote concernant l'idée de système précède Wolff, qui la reprend et la subordonne à celle d'Euclide (Voir l'introduction de M. Albrecht dans son édition de (Wolff [1729] 2019, p. 12-13)). Nous revenons sur l'histoire de ce terme plus loin section 7.6.
94. (ibid., p. 40-41).
95. (Barnes 2000, p. 55).
96. (Aubenque [1962] 1972).
97. Voir à ce sujet (Crubellier et Pellegrin 2002, p. 229).
98. (Barnes 2000, p. 56).

Dans ce débat, il semble qu'il faille renverser les perspectives : les sens contemporains de *système*, dont nous allons plus loin donner une courte généalogie, sont eux-mêmes des inventions des Temps modernes et, comme nous l'avons dit, ils se cristallisent en partie *en référence à* la théorie de la science exposée dans les *Seconds analytiques*. C'est cela qui nous permet d'utiliser le terme dans un commentaire de ce texte, tout en prenant garde à ce qu'il n'y infiltre pas, comme par contrebande, les sens qu'il véhicule pour nous aujourd'hui.

De fait, pour Barnes, les *Seconds analytiques* sont un « traité sur la méthode axiomatique – il ne s'agit pas de développer une science mais plutôt d'examiner la manière dont une science devrait être développée[99]. » Ils ne présentent aucunement une doctrine de l'investigation scientifique, mais seulement des préceptes concernant l'exposition systématique et la didactique de la science une fois celle-ci constituée et la recherche achevée[100]. Barnes, qui écrit du point de vue de la philosophie analytique des sciences, a peut-être à l'esprit la distinction entre *contexte de la découverte* et *contexte de la justification*. Et pour les positivistes logiques, la logique n'a en effet affaire qu'à la forme achevée du système scientifique. Dans le modèle dit « classique » de l'explication scientifique, dû à Hempel (1965), celle-ci procède bien par déduction de lois à partir de lois plus fondamentales, notamment par l'introduction de termes intermédiaires qui permettent de spécifier ces dernières. Elle a donc bien pour forme le syllogisme, ainsi que l'affirme Aristote, notamment dans sa modalité affirmative universelle, si on prend garde de s'intéresser ici à *la forme de son argumentation* plutôt qu'à l'enchaînement détaillé des calculs.

Pierre Pellegrin apporte une vision plus nuancée sur ce que pourrait signifier la systématicité chez Aristote. Il écrit, dans la présentation liminaire de sa traduction des *Seconds analytiques* :

> Ce mode éminent de savoir [qu'est la science démonstrative] prend la forme d'un système déductif fondé sur des principes vrais, antérieurs aux conclusions qu'on en tire et causes de ces conclusions, entraînant chez celui qui le possède une conviction inébranlable[101].

Dans son introduction, Pellegrin affirme que c'est cette dernière propriété qui est

99. (Barnes 2000, p. 57).
100. Voir (Barnes 1969), ainsi que l'introduction de P. Pellegrin à son édition des *Seconds Analytiques*, p. 9-10.
101. P. Pellegrin, édition des *Seconds Analytiques*, quatrième de couverture.

essentielle :

> Les *Seconds Analytiques* contiennent assurément la description d'un système du savoir que l'on peut brièvement caractériser comme un corps de propositions mises sous une forme déductive et axiomatisée. […] Mais là n'est sans doute pas l'essentiel du projet qu'Aristote déploie dans les *Seconds Analytiques*. Ce que, en effet, il entend d'abord y définir, c'est un certain *type de connaissance*, la science entendue comme « science démonstrative » que « nous possédons par le fait de posséder une démonstration »[102].

En d'autres termes, c'est bien la science (ἐπιστήμη) en tant que disposition ou vertu intellectuelle qui est essentielle ici[103] c'est-à-dire, plutôt que le système de propositions lui-même, la capacité à savoir *systématiquement* ce que sont les choses, pourquoi elles sont telles, etc. On rejoint ici l'insistance de Wolff lui-même sur l'esprit systématique plutôt que sur le système, visible dans la citation donnée plus haut.

Ainsi Pellegrin accorde à Barnes que ce n'est certes pas de la découverte scientifique que parlent les *Seconds analytiques*, mais il conteste qu'Aristote ait eu à l'esprit nos considérations formelles contemporaines. Ce qu'il vise à expliciter, c'est la « connaissance qu'un sujet humain peut avoir des connexions nécessaires qui se trouvent dans le monde[104] », et qui le rend capable d'expliquer les phénomènes ou de les prédire de manière assurée.

Il est peut-être possible d'aller un peu plus loin dans cette direction, et de réhabiliter la valeur heuristique des *Seconds analytiques*. En anticipant quelque peu sur nos prochaines réflexions, il faut remarquer qu'on peut entendre deux choses par *investigation scientifique*. Dans le sens qui est entendu par Barnes et Pellegrin, il s'agit des investigations visant à établir les théories scientifiques elles-mêmes – ce que nous appellerions la recherche fondamentale aujourd'hui. Ce qu'on entend par contre par *résolution de problème* désigne d'autres activités investigatrices, plus répandues et moins spectaculaires, qui consistent à expliquer et à prédire les phénomènes dans le cadre de théories déjà constituées – par exemple pourquoi telle galaxie a telle vitesse, ou ce que peuvent être ces influx lumineux réguliers perçus à tel endroit du ciel, ou encore quand aura lieu la prochaine éclipse de lune. On peut se rappeler ici la distinction de Ryle entre le fait de construire une théorie et

102. Introduction de P. Pellegrin à son édition des *Seconds Analytiques*, p. 48.
103. Pellegrin reprend cette idée à (Brunschwig 1981).
104. Introduction de P. Pellegrin à son édition des *Seconds Analytiques*, p. 47.

celui de l'utiliser pour donner des explications et résoudre des problèmes[105]. Le livre I des *Seconds analytiques* donne les critères et les formes principales d'une « bonne » explication ou d'une bonne résolution de problème dans le cadre d'une théorie scientifique déjà constituée, critères par lesquels elle se distingue essentiellement des résolutions de problèmes techniques et pratiques dont nous avons parlé plus haut.

Il faut notamment qu'il y ait une suite bien ordonnée de connexions entre les prémisses et la conclusion, qui reflète la structure sous-jacente des choses mêmes dont il est question. Il faut que chaque connexion soit nécessaire, c'est-à-dire minimale : elle doit viser exactement la relation *réelle* qui existe entre les termes reliés ; il faut enfin que la solution soit universelle relativement aux cas de figure envisageables ; par exemple elle doit être invariable dans le temps[106]. Ainsi, le scientifique peut répondre exactement aux problèmes posés, avec le minimum de données nécessaires, en couvrant toutes les situations envisagées ; il peut justifier et garantir sa réponse par des enchaînements rigoureux de raisons s'ancrant ultimement dans l'observation correcte du réel.

Ce qui donne ces propriétés aux résolutions scientifiques de problèmes, c'est que justement elles se déroulent au sein d'un système dont les termes premiers et les lois ont été identifiés et formulés d'une manière minimale pour couvrir tous les problèmes possibles. Ce travail plus fondamental *de réduction systématique* est l'objet, entre autres, des considérations d'*APo.* II sur la définition. En effet, comparativement au livre I plutôt tourné vers la démonstration explicative, qui est pour Aristote la forme principale du problème scientifique, ce second livre nous semble orienté vers la constitution du système scientifique lui-même. En particulier, le chapitre 13 décrit l'exigence de constituer et d'organiser des listes *exhaustives* de termes descriptifs quadrillant le domaine en question. La méthode platonicienne de la division, dont Aristote est par ailleurs critique, nous semble ici tenir lieu d'un modèle simplifié dont le mérite est de donner l'idée d'une démarche exhaustive arrimée à l'observation du réel. Le chapitre 8, quant à lui, montre qu'il existe des termes qui peuvent être dérivés d'autres termes plus fondamentaux par la démonstration. Une fois qu'on a en effet l'explication du *pourquoi* d'une éclipse ou du tonnerre, on a *en même temps* leurs définitions, et celles-ci ne doivent donc

105. Voir plus haut, § 92.
106. Ces propriétés sont exposées notamment en *APo.* I.2 et *APo.* I.4.

plus figurer parmi les termes élémentaires de la science[107].

Cela nous ramène au *Pont aux ânes* et à son application possible à l'astronomie. Aristote, de manière significative, ne parle plus dans le passage cité plus haut de *listes de termes*, mais des *principes* qui permettent de les générer systématiquement. Une science achevée est une discipline qui fournit *par avance* les termes nécessaires à l'application du *Pont aux ânes*– ou plus généralement, dirions-nous aujourd'hui, à l'application de méthodes procédurales pour la résolution de problème. Dans un tel cadre, une méthode comme le *Pont aux ânes* devient envisageable *en droit*, car l'espace des solutions possibles est délimité par le nombre fini des termes premiers disponibles.

Il s'agit là d'un second sens qu'on peut donner à l'idée que le scientifique a la capacité de résoudre *systématiquement* des problèmes, et cette fois-ci il est bien question de procédure et d'heuristique. On peut certes s'étonner de la naïveté du dispositif d'Aristote quand on considère la complexité des raisonnements scientifiques d'aujourd'hui. À cela on peut d'abord répondre que cette naïveté fut partagée par Leibniz et, au moins un temps, par Simon qui donna le titre ronflant de General Problem Solver (GPS) à un algorithme qui n'est rien d'autre qu'une version récursive du *Pont aux ânes*[108]. Mais au-delà de l'insuffisance des moyens logiques dont dispose Aristote, il y a bien là l'idée fondamentale, qu'à un problème suffisamment circonscrit par un système scientifique doit pouvoir correspondre une méthode de résolution consistant dans une certaine manière d'explorer ce système à la recherche de solutions. Comme le dit Crubellier (déjà cité), ce qui intéressait Aristote dans le *Pont aux ânes* c'est que de telles procédures systématiques soient possibles *en théorie*, c'est-à-dire qu'on puisse être assuré de trouver une solution en parcourant selon une procédure systématique l'espace quadrillé par le système scientifique. Car alors une autre démarche de résolution de problème devient envisageable, celle que promeut Simon après la déception provoquée par le GPS, de développer des « stratégies de recherche » pour réduire progressivement cette complexité[109] ; c'est ce que font de nombreux algorithmes modernes, en réduisant progressivement l'espace des solutions possibles d'un problème donné, et c'est ce que fait déjà le *Pont aux ânes* de manière modeste, en classant les listes de termes en groupes ayant de fortes incompatibilités entre eux ainsi que des priorités

107. *APo.* II.8, 93b4-14.
108. Voir plus haut, § 64, et annexe A.
109. Voir plus haut, § 65.

différentes de considération. Aristote note avec satisfaction cette propriété de sa méthode, puisqu'elle nous dit comment « chasser les modes des syllogismes, sans avoir à inspecter tous les prédicats[110] ».

§ 106. L'unité de l'investigation et ses espèces

De cette enquête se dégage une nouvelle perspective sur la nature de l'investigation (*ζήτησις*). Si elle est bien commune aux sciences, aux techniques et à la pratique, elle s'y présente cependant sous des aspects bien différents. La délibération constitue l'*essentiel* du savoir dans le domaine éthique, où il faut à nouveau délibérer dans chaque cas particulier. Comme le dit Allen[111], elle détermine certes un théorème, mais ce théorème est « à usage unique », adapté seulement à la situation rencontrée ; le raisonnement devra être réitéré à chaque fois, ne serait-ce que pour s'assurer de sa similarité avec la situation ayant servi de modèle. Le cas des techniques est intermédiaire ; elles visent à s'affranchir de la délibération, mais se contentent souvent de la faciliter en l'encadrant fortement.

Dans le système achevé de la science, l'investigation est doublement destinée à disparaître, d'abord parce que l'homme de science aura atteint sa fin, la contemplation des principes, et ensuite parce que, s'il doit parfois encore résoudre des problèmes particuliers, il disposera alors du système de vérités essentielles lui permettant, de manière procédurale, de trouver la réponse à toute question. C'est ce qui explique peut être qu'Aristote, occupé à décrire cet état final du savoir dans les *Seconds analytiques*, en ait si peu parlé.

Cependant, dans la science réelle et inachevée, l'investigation est bien présente à travers l'opposition forte qu'Aristote institue entre l'état de savoir en acte et le mouvement d'acquisition du savoir. Cela se voit bien dans la formulation d'*APo*. II.1 et sa construction en miroir :

> Les choses que l'on cherche sont en nombre égal à celles dont on a un savoir scientifique.
>
> Τὰ *ζητούμενά ἐστιν ἴσα τὸν ἀριθμὸν ὅσαπερ ἐπιστάμεθα*[112].

Ainsi, la distinction entre raison épistémique et raison délibérative en *Éth. Nic.* VI.1 semble en cacher une autre, tout aussi fondamentale, qui opposerait raison épistémique et raison *investigatrice* (pour traduire

110. *APo*. I.30, 46a12.
111. Voir plus haut, p. 394.
112. *APo*. II.1, 89b22.

« zététique »). Cette dernière inclurait la raison délibérative des affaires pratiques et techniques, mais également la réflexion qui est à l'œuvre dans les problèmes scientifiques.

Cette unité de l'investigation est évoquée de manière discrète, nous semble-t-il, dans un texte du *De anima* qui décrit le passage de la science en puissance à la science en acte. Ce passage est décrit comme un mouvement de la faculté noétique (*τὸ νοητικὸν*) qui poursuit le bon et évite le mauvais, et « pour ce qui est sans action, le vrai et le faux, qui est du même genre que le bon et le mauvais[113] ». On peut comprendre ce texte elliptique comme une allusion à l'investigation (*ζήτησις*) scientifique qui, comme « on calcule et on délibère des choses futures à partir des choses présentes[114] » conduirait sa recherche en explorant les pistes semblant avoir le plus de vraisemblance, et en évitant celles qui en auraient moins. Et c'est à cette raison investigatrice qu'auraient affaire les dispositifs procéduraux des *Topiques* et des *Premiers analytiques*.

6.4 Système technique et contingence

Nous commençons par approfondir le lien que nous avons aperçu entre les notions de *système* et de *résolution de problème*. Le système permet d'être exhaustif car il donne la possibilité d'être *parcouru* entièrement à la recherche d'une solution. On peut voir là le cœur de l'idée de *méthode procédurale* qui ne résout pas ses problèmes grâce à sa pénétration de leur essence, ce qui requiert « vivacité d'esprit » (*ἀγχίνοιά*), dit Aristote, mais par une démarche comme extérieure de recherche dans un espace de solutions (§ 107). Nous montrons ensuite en quoi Aristote pouvait à juste titre être sceptique quant à la possibilité de constituer des systèmes techniques répondant à un tel cahier des charges, en revenant sur l'opposition *nécessaire / contingent* (§ 108). Nous exposons cependant pourquoi une telle possibilité reste néanmoins pensable dans un cadre aristotélicien, ce qui permet de cerner au plus près, pour finir, la nature de l'opposition entre raison théorique et raison pratique que nous a léguée le Stagirite (§ 109).

113. *De anima* III.7, 431b10
114. *De anima* III.7, 431b7

§ 107. L'esprit de méthode contre la vivacité d'esprit

La solution d'un problème donné est systématique, avons-nous dit, si non seulement elle est correcte et qu'on peut la démontrer, mais également si elle est exhaustive relativement aux cas possibles et (de manière optionnelle) si elle est minimale vis-à-vis de la consommation de ressources argumentatives. Or de telles propriétés ne peuvent être montrées que dans le cadre d'un système de référence qui permet d'explorer des options différentes. Dans un passage étonnant d'*APo*. I.33, Aristote soutient que le savoir scientifique ne se distingue pas de l'opinion vraie par ce qu'il serait fondé par une démonstration, et elle non, car il est possible

> que l'homme qui opine procède par la voie du moyen terme jusqu'à ce qu'il parvienne aux termes immédiats […] Car il est possible d'opiner sur le fait aussi bien que sur la raison du fait, *i.e.* le moyen terme[115].

Si de tels raisonnements ne sont pas des démonstrations scientifiques, dit Aristote, c'est qu'ils ne saisissent pas les propositions enchaînées comme nécessaires, et cela à son tour, parce qu'ils ne reconnaissent pas les définitions comme essentielles, c'est-à-dire qu'ils ne reconnaissent pas *où* sont les principes et où sont les accidents, ce qui est nécessaire et ce qui n'est que contingent. En d'autres termes, le raisonnement d'opinion ne dispose pas du « système de coordonnées » qu'est la théorie scientifique, et qui permet de le *localiser*, c'est-à-dire d'expliciter comment il se rattache aux principes et par là, d'envisager des options différentes et de le comparer à celles-ci.

Ce qui permet cette « exploration systématique » de l'ensemble des vérités, est une propriété élémentaire de l'apodictique que présente Aristote en *APo*. I, à savoir qu'elle a une structure *récursive*, comme le note Barnes[116]. On pourrait ajouter qu'elle partage cette propriété avec la diérèse platonicienne qu'elle veut concurrencer. Dans le cas de la diérèse, un terme est soit premier, soit obtenu à partir de la division d'autres termes, nous dit Platon. Dans le cas de la théorie de la démonstration, une proposition vraie est soit un principe, soit une proposition démontrée à partir d'autres propositions vraies, nous dit Aristote :

> Un syllogisme est une démonstration lorsqu'il procède de [prémisses] vraies et premières, ou de [prémisses] dont la connaissance prend naissance de [prémisses] vraies et premières[117].

115. *APo*. I.33, 89a14.
116. (Barnes 1981, p. 26, note).
117. *Top*. I.1, 100a27. Voir également *APo*. I.2 dont les formulations sont moins immédiatement récursives.

Barnes[118] a montré les difficultés qu'a la logique des *Analytiques* à donner à cette structure récursive un caractère infinitaire, *i.e.* qu'elle permette de générer un nombre indéfini de conclusions à partir d'un nombre fini de termes premiers, une propriété qu'il semble pourtant supposer[119]. Cependant, quelles que soient ses difficultés à ce sujet, il nous semble que le point essentiel pour Aristote est que, dans une telle structure, l'exploration peut être systématique car il suffit de partir des premiers termes pour reconstruire, à l'aide des règles de dérivation ou de division, l'ensemble des termes possibles afin de les examiner un à un[120]. C'est l'intérêt porté à établir cette possibilité d'exploration systématique qui pourrait expliquer les nombreux chapitres qu'Aristote consacre à établir les propriétés « topologiques » du savoir scientifique dans les *Seconds analytiques*, comme par exemple la clôture des démonstrations, l'impossibilité de démonstrations circulaires ou infinies, de quelque manière que ce soit, la possibilité qu'il y ait plusieurs démonstrations d'une même proposition, l'exclusion de l'aléa[121].

Ces développements pourraient laisser croire que nous souscrivons à la thèse générale de Barnes, et que nous suggérons de faire d'Aristote un précurseur de l'idée moderne de systèmes formels récursifs. Il suffit cependant à notre propos de montrer que cette idée contemporaine de la systématicité, qui nous intéresse particulièrement étant donné son lien avec l'informatique, *n'est pas incompatible* avec la pensée du Stagirite quant à la forme idéale de la connaissance. La source de l'intuition d'Aristote nous semble résider, comme le dit Pellegrin, dans sa tentative de caractériser l'*état d'esprit* du scientifique, qui le distingue essentiellement de celui de l'homme d'opinion, et qui est d'examiner les problèmes non comme ils viennent, avec leur propre arsenal de concepts et d'hypothèses, mais *à partir* d'un cadre de réflexion qui en donne la formulation et comme les coordonnées exactes selon des termes prédéfinis, et qui oblige leur résolution à procéder selon des voies et des règles pré-établies.

Peut-être voit-on mieux l'« innovation » d'Aristote si on la compare à l'ana-

118. (Barnes 1969, p. 147-150).

119. *APo.* I.32, 88b6. Ce passage est jugé obscur par les commentateurs et interprété de diverses manières.

120. Cela est vrai de la logique propositionnelle, qui est complète et décidable. Dans le cas de logiques plus complexes, l'exploration systématique doit se faire au niveau sous-jacent des théorèmes, celui des expressions valides.

121. Voir respectivement : *APo.* I.7, *APo.* I.3, *APo.* I.19-23, *APo.* I.29, *APo.* I.30. Voir une réflexion similaire de Barnes, qui parle de la « structure » d'un système axiomatique, p. xx de son édition des *Seconds analytiques*.

mnèse de Platon. Contrairement à ce dernier, Aristote ne pense pas qu'il est possible, face à chaque question et chaque problème, d'entreprendre leur résolution en les considérant dans leur particularité propre et en remontant, par la seule puissance de l'analyse de concepts, jusqu'à des notions communes universellement acceptables parce qu'à la source de la rationalité. Une telle conception surhumaine de l'intelligence ne lui semble pas refléter la réalité du savoir, qui progresse plutôt par accumulations locales à chaque discipline. Il intercale ainsi entre l'expérience particulière et les notions communes l'idée de « sciences », c'est-à-dire de dispositions à appréhender un certain domaine de choses et leurs connexions internes dans leur universalité et dans leur nécessité, dispositions qui ne sont pas innées mais qui doivent être acquises et construites patiemment, par la découverte ou par l'apprentissage.

Il serait ainsi possible d'avancer l'hypothèse suivante : Aristote, semble-t-il, aurait bien perçu le défaut de toute méthode dialectique qui tient à ce que, quelle que soit la sophistication de ses procédures internes d'argumentation, son effectivité dépend ultimement de la qualité des données fournies initialement. Le problème des listes canoniques de termes nécessaires au bon fonctionnement du *Pont aux ânes* ne nous semble pas avoir été ignoré naïvement par Aristote : au contraire le texte cité des *Top*. I.14[122] montre qu'il en avait l'amère et laborieuse expérience. Par la théorie du syllogisme il conçoit, comme le montre Crubellier, une matrice universelle d'argumentation par laquelle il entrevoit la possibilité de réduire ces listes *ad hoc* à celles, uniques pour chaque science, des termes élémentaires dont tous les autres pourraient être dérivés. Ainsi la démarche procédurale des *APr.* I.27-30, et plus généralement de toute la théorie du syllogisme, nous semble entrer en résonance avec la doctrine de la science exposée par les *Seconds analytiques justement* par cette idée de *résolution systématique de problème*. Une science, comme système de concepts et de principes, a certes besoin de règles de dérivation univoques comme celles du syllogisme pour réduire la profusion des concepts à un ensemble minimal de définitions premières. Mais la résolution de problème selon ces règles requiert à son tour, pour être tout simplement effective, de pouvoir s'exercer au sein d'un tel système préétabli.

En deçà donc de la forme d'une axiomatique déductive à laquelle Barnes réduit toute l'idée de système, il nous semble que l'intuition d'Aristote est plus originaire. Vouloir résoudre un problème « méthodiquement » implique nécessairement de le

122. Voir plus haut, p. 415.

considérer, au moins sous un certain aspect, comme un problème *quelconque* sans se préoccuper de sa particularité. On s'intéresse par exemple à la forme de la question, aux termes qu'elle emploie, aux moyens qui sont mis à disposition, à l'état d'esprit du requérant, etc. et, à partir de ces caractéristiques *externes* au problème lui-même on agit de telle ou telle manière générique. Les méthodes proposées par les *Topiques* et le *Pont aux ânes* sont caractéristiques à cet égard. Elles n'abordent pas le sujet en allant « droit au but » et en faisant appel au sens commun, mais elles l'encerclent progressivement par des observations, des collectes auxiliaires de faits, des rapprochements avec des schémas préétablis. En ce sens elles peuvent paraître rhétoriques et dialectiques, et dévoyer l'esprit même de la dialectique telle que l'avait établie Platon c'est-à-dire de viser directement la chose même dans son essence.

La méthode de la diérèse, pratiquée par le Socrate de Platon, est chaque fois un miracle de l'intelligence. Pratiquée par de moins habiles élèves, elle n'est qu'une vague recette aux résultats incontrôlables. Aristote dégage l'idée de systématicité en prenant acte de ce défaut de toute méthode, qu'elle peut aisément être dévoyée en formules répétitives et insensées, voire détournée à des fins sophistiques : elle ne saurait pallier seule les défauts d'intelligence ou de probité de l'enquêteur. Il n'en tire pas cependant la conclusion que cette voie méthodique doit être rejetée. Il « double la mise » au contraire en dégageant l'idée que toute méthode, afin d'être effective, doit elle-même être insérée dans un système de connaissances, vérifié par ailleurs, qui la contraint. Dans un tel système, on continue d'aborder un problème *de l'extérieur*, mais il ne s'agit plus du tout de la même chose. Loin de s'appuyer de manière arbitraire sur des caractéristiques accidentelles du problème à résoudre, la méthode l'enserre ici tout entier au sein du système comme une battue de chasse rabat ses proies dans un cercle toujours plus étroit de la forêt, pour prolonger une image de l'investigation qu'affectionne Aristote[123].

Derrière l'idée du *Pont aux ânes* et, pensons-nous, plus généralement de méthodes procédurales, il y a celle toute simple d'une « recherche systématique » telle qu'on l'entend dans le langage courant, lorsque par exemple, désespérant de trouver un objet aux endroits usuels, on décide de ne plus faire d'hypothèse quant à son emplacement et d'explorer les pièces de la maison une par une, dans leurs moindres recoins. Se défier ainsi de l'intuition et de la « vivacité d'esprit » pour

123. Barnes (1969, p. 147) note que le terme *chasse* (*θήρα*) apparaît quatre fois dans les *Analytiques*.

trouver le moyen terme[124] n'est pas seulement un pis-aller pour les lourdauds, comme nous l'avons vu. La procédure, toute pédestre qu'elle soit, donne la garantie que la solution se trouve dans un certain périmètre délimité par le système. Elle permet de démasquer les fausses intuitions, de répertorier toutes les autres possibilités, d'identifier la meilleure d'entre elles, et souvent de résoudre d'un seul coup toute une classe de problèmes similaires. Les grandes oppositions que nous avons brossées à grand trait, au chapitre précédent, entre les écologies cognitives humaines et les procédures industrielles et informatiques se trouvent d'une certaine manière déjà contenues dans la tension implicite et irrésolue entre les deux voies pour trouver le moyen terme qu'indique Aristote à près de quarante pages d'intervalle dans les *Analytiques*, le *Pont aux ânes* et la vivacité d'esprit, sans que nulle part il se résolve à les hiérarchiser.

§ 108. Contingence et nécessité

Cette idée préliminaire de la systématicité étant dégagée, on peut revenir à présent à notre question principale concernant la possibilité pour les techniques de se doter de méthodes procédurales. La réponse d'Aristote décrite aux sections précédentes se précise donc ainsi : même si le *Pont aux ânes* leur est ouvert en droit, comme à toute autre discipline, une telle procédure sera de peu de profit aux techniques puisqu'elles ne peuvent pas espérer, contrairement aux sciences, enserrer leurs concepts au sein d'une totalité systématique. Elles devront établir laborieusement, pour chaque problème donné, des listes de termes décrivant le problème à résoudre. Comme nous l'avons vu plus haut avec l'exemple de la rhétorique, la technique permet au mieux de cadrer cette activité préliminaire de description du problème, mais nullement de se substituer à elle.

Or les raisons pour lesquelles Aristote pensait que les techniques ne peuvent se constituer en systèmes déductifs ne sont plus du tout claires pour nous. Aristote semble lui-même hésiter sur le sujet, puisqu'en *APr.* I.30, après avoir mentionné la réduction des termes aux principes en astronomie, il affirme qu'on peut faire de même « en tout art et en toutes science ». Cependant, dans les *Seconds analytiques*, il semble bien réserver la forme systématique aux seules sciences. En effet, il ne peut y avoir ni principes ni démonstration des choses contingentes :

> Il existe certaines choses qui bien que vraies et existant réellement peuvent être autrement. Il est donc clair que de telles choses il n'y a pas de science […] ni d'intel-

124. *APo.* I.34, 89b10.

> lection (j'appelle intellection un principe de science), c'est-à-dire d'une science non démonstrative, qui est la saisie d'une prémisse immédiate[125].

Jusqu'ici nous avons accepté telles quelles les oppositions établies par Aristote entre science et technique, selon lesquelles la première avait affaire aux causes réelles des choses et à leurs principes nécessaires, tandis que la seconde se contentait d'identifier les moyens généraux de l'action plongée dans des situations contingentes.

Ces oppositions, assez acceptables intuitivement, peuvent justifier que science et technique soient rapportées à deux facultés différentes de l'âme. Cela n'explique cependant pas pourquoi, visant toutes deux l'universel dans leur domaine respectif, elles ne pourraient pas toutes deux également s'organiser en système déductif autour de principes de raisonnement et de concepts primitifs. Le succès des « sciences de l'ingénieur » à notre époque montre assez clairement, semble-t-il, la fausseté d'une telle hypothèse.

Il nous faut donc revenir aux deux oppositions entre cause et moyen d'une part et entre nécessité et contingence de l'autre, afin de comprendre pourquoi Aristote les associe, semble-t-il à tort, à cette troisième entre l'esprit systématique et l'esprit non systématique. En particulier, la dépendance envers des principes nécessaires et vrais semble assez accidentelle à notre idée moderne de *système*. Sa valeur épistémique réside plutôt, pour nous, dans la validité de ses connexions internes, dans son « auto-portance logique », aimerions-nous dire, qui reste intacte quand bien même ses principes initiaux (axiomes, définitions) ne seraient pas acceptées. On parle en ce sens de *système hypothético-déductif*. Or ce mode conditionnel de vérité est justement, avons-nous dit à la section précédente, celui de la technique : il semble donc que l'idée de *système* lui soit très adaptée.

En *Mét.* Δ.5, Aristote envisage quatre manières de comprendre le terme de *nécessité*, qu'il subsume sous la définition générique de « ce qui ne peut être autrement qu'il n'est[126] ». Nous pouvons mettre de côté la nécessité logique qui lie les conclusions aux prémisses, puisque c'est justement là celle qui fait l'unité du système déductif, dont nous cherchons les objets possibles. Outre la nécessité des êtres parfaits et simples, qui semble se référer à la théologie (une science non démonstrative) Aristote mentionne deux formes de nécessité conditionnelle : celle des causes efficientes, qui contraignent le mouvement naturel d'une chose, et celle

125. *APo.* I.33, 88b33.
126. *Mét.* Δ.5, 1015a34.

des causes finales, qui requièrent des moyens afin d'être réalisées. C'est donc sur ces deux formes que peuvent porter les sciences démonstratives.

Notre instinct moderne associerait naturellement la science au savoir des connexions efficientes entre les choses, et la technique à celle des connexions finales. Ces dernières devant ultimement être rapportées à des intentions humaines qui sont contingentes, elles ne seraient donc pas objet de science. Or une telle interprétation n'est que partiellement exacte d'un point de vue aristotélicien. Aristote en effet, dont la science de la nature est de part en part fonctionnaliste, voit plutôt dans les connexions finales le règne de la vraie nécessité, tandis que les connexions efficientes sont le règne de l'aléa[127]. C'est la dynamique interne de transformation des formes naturelles qui explique les mouvements externes, qui sont violents *relativement à une autre forme naturelle* dans la mesure où ils viennent contraindre sa propre dynamique. Les phénomènes mécaniques sont par essence le lieu du hasard[128], c'est-à-dire de l'interférence accidentelle de séries causales (finales) par ailleurs parfaitement explicables. Cependant, lorsque ces interférences sont régulières parce qu'elles concernent des principes nécessaires, comme certains phénomènes météorologiques qui sont dus aux mouvements des astres, elles peuvent devenir par extension objet de science. Deux catégories d'êtres en sont par contre exclus : d'une part, les phénomènes mécaniques naturels irréguliers (comme un tremblement de terre), et d'autre part tout ce qui a trait aux intentions humaines contingentes.

On appelle ces deux formes de contingence *fortune* (*τύχη*) *en tant* qu'elles perturbent des intentions humaines. Elles limitent les ambitions des techniques de manière différente. Les aléa mécaniques bornent leurs ambitions de fiabilité et de précision, mais là n'est pas l'essentiel pour notre propos. C'est la diversité et la volatilité des intentions humaines qui empêchent les techniques de se constituer en systèmes.

Si on quitte en effet notre point de vue contemporain, pour se placer du point de vue de l'observateur traditionnel des arts et métiers – point de vue qui est celui des *réductions en art* des Temps modernes[129] et même encore celui des encyclopédistes – on peut comprendre ce qu'a de plausible une telle position. Certes, l'atelier

127. *Phys.* II.8. Voir les deux études de Lennox (1982, 1984) sur la notion de hasard chez Aristote.
128. Le terme ici employé en grec soit *αὐτόματον*. Voir (Presti 2012, p. 26-12) pour les différents sens de ce terme et leur connexion conceptuelle en Grèce ancienne.
129. Voir plus loin § 112.

d'un artisan, par exemple une forge, peut impressionner par son organisation, son savoir-faire, ses procédures, et on peut sans doute décrire tout cela de manière systématique à l'échelle locale. Mais les manières de faire spécifiques à cet atelier dépendront d'une part des traditions, de l'organisation du travail, de la forme des outils et de la qualité spéciale des matières ayant cours dans la région, et d'autre part de la diversité des besoins et des habitudes des clients, qui orienteront de manière décisive les activités de l'atelier. En d'autres ateliers et d'autres régions, on trouverait de nombreuses similarités concernant l'organisation et les procédés généraux de la fabrication, mais sans doute autant de différences de détail, qui iraient se composant avec la multiplication des instances d'observation. L'universalité de la description irait s'appauvrissant, et ne serait sans doute pas capable d'organiser les variations observées en une taxonomie cohérente, leurs facteurs pouvant être combinés de manière sans doute arbitraire – les écologies cognitives humaines introduites au chapitre précédent sont justement faites d'ajustements multiples et réciproques entre les pratiques humaines et leur environnement matériel et socioculturel. Se désintéressant des causes et s'arrêtant, dans leur remontée vers l'universel, au premier système stable et efficace de moyens qu'elles rencontrent, les techniques ne sont capables que de *maxima locaux*, pour prendre une image mathématique.

La technique n'est donc pas incompatible *en droit* avec l'idée de *système* parce que ses principes seraient contingents, mais elle l'est *de fait* parce qu'ils sont trop variables et qu'ils ne sont pas du type de « ce qui arrive le plus souvent » ce degré appauvri de nécessité qu'Aristote admet néanmoins parmi les modalités de la science de la nature[130]. Ou bien, pour être plus précis encore, ils n'exhibent qu'une stabilité locale, qui ne s'élève que médiocrement à l'universel. C'est un autre sens en lequel on peut entendre l'image de l'échelle des savoirs déjà commentée plus haut : les systèmes de savoirs techniques, s'il devait y en avoir, seraient locaux et modestes – et sans doute pas d'une grande utilité.

§ 109. La possibilité de systèmes techniques

On peut résumer les résultats de ce chapitre de la manière suivante : La distinction *essentielle* que fait Aristote entre la connaissance théorique et la connaissance pratique se justifie par leurs *objets* différents. La première porte sur la connaissance

130. *APo.* I.30, 87b25, *Phys.* II.8, 198b35. Voir les commentaires de Barnes dans son édition des *Seconds analytiques*, p. 192 et de (Mignucci 1981) sur cette notion très discutée.

de *ce qui est*, et la seconde de *ce qu'il faut faire*. C'est de cette distinction essentielle que procèdent toutes les autres, y compris celles qui existent entre science et technique, qui pourtant toutes deux peuvent prétendre à l'universalité :

1. Les problèmes et les systèmes de la technique sont radicalement contingents en ce qu'ils procèdent des *intentions humaines rationnelles*, qui *voient* les choses non telles qu'elles sont, mais telles qu'elles *pourraient être*. Elles ne portent donc pas sur des êtres naturels (objets ou situations) mais sur des transformations possibles de ces êtres, sur des artefacts.
2. Les savoirs techniques portent sur des délibérations et des méthodes, et non pas sur des faits ; ils peuvent mobiliser des faits, mais ne s'y réduisent pas ; ils traitent du *comment*, et non du *quoi*.
3. L'attitude technique ne vise pas à découvrir les causes premières des choses, mais les moyens les plus efficaces de l'action ; leur critère de validité n'est pas la vérité, mais la réussite. Ils sont en cela pragmatiques et, s'ils peuvent mobiliser les savoirs scientifiques, ils s'arrêtent dans cette démarche à l'horizon borné par leur intérêt.

La troisième distinction se développe dans celle, plus célèbre encore, que dresse Aristote entre vie active et vie contemplative en *Éth. Nic.* X.7. Cette dernière est jugée supérieure à la première en ce qu'elle vise à comprendre les choses mêmes, de manière désintéressée, en ne se préoccupant de les saisir que selon leur essence et selon leurs causes. Au contraire, l'esprit délibératif ne s'intéresse aux choses qu'en tant qu'elles servent ses fins propres. Il y a donc une différence d'intention radicale de l'acte de connaissance qui, selon Aristote, justifie qu'on les distingue *génériquement*. Il est important de remarquer à cet égard que, dans ce texte d'*Éth. Nic.* X.7, les vertus épistémiques auxquelles fait référence Aristote pour caractériser la vie contemplative sont l'intellect (*νοῦς*), principalement[131] et la sagesse théorique (*σοφία*)[132], plutôt que la science (*ἐπιστήμη*) qui n'est pas citée. Celle-ci participe donc certes à la vie théorétique, mais seulement en tant qu'elle permet d'accéder aux premiers principes, saisis par le l'intellect. En d'autres termes, ce n'est pas le « labeur » de l'investigation (les expériences, les démonstrations, les applications) qui intéresse la vie théorique, mais bien la saisie de ce qui se trouve au fondement de la réalité physique et mathématique – la nature de la matière, de l'énergie, du nombre, du Bien, etc.

131. 1177a14, 21, b30.
132. 1177a24, 34.

Thomas d'Aquin résumera toutes ces distinctions en disant que connaissance spéculative et pratique se distinguent selon leur objet (être naturel ou artefact), selon leur modalité (connaître ce qu'est la chose ou comment la faire), et selon leur fin (connaître pour la vérité ou pour l'utilité)[133].

À ces trois distinctions, nos dernières réflexions en ajoutent une quatrième, entre la forme systématique de la science, qui est possible parce que le processus de recherche des causes converge vers des principes premiers, et celle seulement « rhapsodique » de la technique, dont les moyens n'ont qu'une portée locale, et qu'une tentative de trop généraliser déliterait dans une explosion de circonstances et de cas particuliers.

Nous souhaitons ici montrer que cette quatrième distinction n'est pas essentielle à la doctrine aristotélicienne, et qu'un changement radical du regard pour penser les pratiques humaines de manière systématique et « à grande échelle » y est possible. Mais il ne s'agit justement pas pour autant d'être plus « scientifique » et de les regarder avec plus d'attention et de déférence encore. Quand on observe « scientifiquement » les pratiques humaines, on reste dans le domaine des sciences humaines – économie, sociologie, anthropologie – qui exhibent bien les difficultés épistémiques que nous venons d'exposer.

Ce changement du regard, qui ne s'est fait jour que progressivement, consiste à ne pas se contenter d'observer les pratiques humaines, mais au contraire à les re-concevoir à neuf et de fond en comble, de sorte qu'elles s'ajustent à l'idée de *système* plutôt que le contraire. Le système n'est pas ici ce qui doit refléter la structure causale interne de ses objets, mais ce qui doit au contraire les organiser et les disposer de manière rationnelle.

Ce que n'aurait donc pas vu Aristote (sans pour autant l'interdire), c'est que l'attitude systématique n'est pas le privilège de l'investigation théorique, mais qu'elle peut également convenir à certaines délibérations pratiques. *Elle est une modalité possible de l'investigation en général*, qui trouve à se réaliser lorsque les savoirs qu'elle accumule peuvent être organisés en système. Cependant organiser des délibérations générales concernant l'action ne peut se faire comme on organise des explications générales concernant les phénomènes naturels. Là on peut espérer remonter aux causes premières dans l'explicitation des principes. Ici, face à la diversité des conditions de possibilité de l'action (finalités, moyens, contraintes, etc.) il faut faire un choix, standardiser en somme, exercer une pression nouvelle

133. Voir plus loin, § 111.

sur l'environnement *extérieur* des pratiques afin que leur logique *intérieure* puisse devenir systématique. En d'autres termes encore, être systématique en technique, ce n'est pas d'abord organiser son *savoir* en système, mais organiser *les situations elles-mêmes* – de sorte que des délibérations systématiques puissent avoir lieu.

Cette possibilité ne remet nullement en cause la distinction radicale que fait Aristote entre science et technique, au contraire. Loin d'affaiblir la thèse aristotélicienne, l'idée moderne que la technique est aussi bien (voire mieux) capable de systématicité que la science, permet en fait de mieux en comprendre la pertinence, *comme au second degré*. Certes, les deux formes de systèmes permettent alors à l'investigation d'élaborer des stratégies procédurales de résolution de problème. L'investigation des causes des choses dans le cadre d'une théorie scientifique ressemble étrangement à la recherche des moyens dans le cadre d'un système technique. La dynamique elle aussi est identique. Dans les deux cas, les solutions trouvées peuvent être ajoutées au système si elles sont jugées intéressantes. Dans les deux cas, la découverte d'une impossibilité amène la remise en cause du système, soit que la théorie soit jugée fausse, soit que le système technique soit jugé insatisfaisant.

Mais il suffit de considérer leur justification respective pour saisir leur différence radicale. Tandis que les systèmes scientifiques visent à donner accès à la compréhension des « connexions nécessaires entre les choses », pour reprendre l'expression de Pierre Pellegrin, les systèmes techniques sont des constructions effectives qui prennent la forme de cahiers des charges adressés à l'environnement qui doit les accueillir – comme notamment la description des comportements attendus de la machine et de ses interlocuteurs au sein de la situation est un pré-requis à la spécification du programme[134]. Tandis qu'un système scientifique permet *par accident* de résoudre des problèmes, parce qu'il vise d'abord les principes, un système technique tire toute sa justification de l'étendue et de l'importance des problèmes qu'il résout, ainsi que de son efficacité relative vis-à-vis d'autres solutions possibles. Il leur est entièrement *contingent*.

C'est dans ce renversement du regard que se situe la contingence radicale des savoirs techniques. Elle ne relève pas du hasard qui afflige les choses du monde sublunaire, mais de cette capacité de l'esprit à *voir les choses autrement*, et cela non seulement quant aux chaînes de moyens que nous savons agencer pour parvenir à nos fins, mais plus profondément encore, quant aux systèmes et aux infrastructures

134. Voir plus haut, § 57.

qu'il faut mettre en place pour maximiser leur disponibilité et leur convenance. La technique a ce pouvoir de démultiplier le monde naturel en mondes intentionnels qui sont, la plupart du temps, les vrais lieux que les personnes habitent[135].

135. Voir plus loin notre lecture d'Anscombe, section 8.2.

Chapitre 7

Machine et système aux Temps modernes

§ 110. Introduction

Au chapitre précédent, nous avons déterminé que, pour Aristote, la technique était la connaissance de l'universalité de certaines délibérations dans une discipline pratique donnée. Nous avons montré en quoi elle pouvait avoir des échanges fructueux avec la science, quoiqu'elle s'en distinguât essentiellement. Nous avons cependant identifié une sorte d'« angle mort » dans la théorie de la connaissance aristotélicienne : alors que ses *Topiques* et ses *Premiers analytiques* déclarent étendre leur portée sur les sciences aussi bien que sur les arts, ses *Seconds analytiques* réservent la forme systématique du savoir aux seules premières. Les raisons d'une telle restriction nous sont parues pertinentes pour les techniques traditionnelles, cependant nous avons développé, en fin de chapitre, une sorte de « déduction *a priori* de la possibilité » de techniques systématisées (en visant bien entendu les notions d'industrie et d'informatique dégagées au chapitre 5) dans un cadre aristotélicien.

Dans ce chapitre, nous reprenons les deux questions directrices de cette partie de notre réflexion. Concernant la résolution systématique de problèmes, on doit d'abord constater l'oubli rapide du *Pont aux ânes* à partir du XVI^e^ siècle, qui fut sans doute lié au discrédit de la logique scolastique[1]. L'idée de systématicité va être ré-élaborée très progressivement au cours des Temps modernes (1500 - 1800), et cela d'abord à partir des concepts de *système* ou *méthode*, termes qui (ré-)appa-

1. (I. Thomas 1965).

raissent à cette période. C'est cependant sur l'idée de *machine*, qui est d'abord un objet géométrique, que les modernes fondent leurs espoirs pour étendre la rigueur des démonstrations mathématiques à l'ensemble des domaines de la connaissance, aussi bien théoriques que pratiques.

En mettant ainsi en évidence les bouleversements conceptuels qui transforment nos idées de connaissance, de science et d'art à cette époque, nous préparons notre analyse de la conception kantienne de l'unité de la connaissance théorique et pratique, et de son opposition frontale à Aristote sur ce sujet : il s'agit d'éviter tout anachronisme dans cette comparaison, tant les idées même de théorie et de pratique ont été reconfigurées dans le temps long qui sépare ces deux philosophes.

La première section 7.1 de ce chapitre est essentiellement historique : elle montre que de nombreux déplacements conceptuels concernant le couple théorie / pratique ont lieu dès le Moyen Âge : nous prenons les thèses de Thomas d'Aquin à ce sujet comme des marqueurs de ces déplacements ; nous rappelons également le phénomène historique des *réductions en art* qui se développe à la Renaissance. Incarnant l'entreprise humaniste de renouvellement et de progrès des savoirs, il trouve son expression réflexive (entre autres) dans l'émergence de l'idée de *méthode* chez Pierre de La Ramée.

Dans la section 7.2, nous nous interrogeons sur l'ambiguïté de la notion de *règle* telle qu'elle apparaît dans les réductions en art : d'une part la règle guide la pratique et l'élève au-dessus de la routine (ce qui peut nous apparaître paradoxal) mais d'autre part aussi, de manière plus conforme à l'idée qu'on s'en fait aujourd'hui, elle permet de faire l'économie du jugement et évite les délibérations inutiles. Nous examinons une résolution possible de cette ambiguïté entrevue par Leibniz, celle d'une domination absolue de la règle en ce second sens – telle qu'elle apparaît dans le calcul. Cependant, en deçà de cette généalogie habituelle de l'informatique (Leibniz - Boole - Babbage - Turing)[2], nous tentons de montrer (section 7.3) qu'il y a d'autres évolutions conceptuelles plus largement déterminantes pour l'avènement d'une technique systématisée, qui se rapportent à une nouvelle division du travail intellectuel, et qui sont centrées sur la figure moderne de l'ingénieur. Les idées de *machine* et de *système technique* jouent un rôle pivot dans la cristallisation d'une telle division du travail.

Après ces études qui concernent la technique, les deux sections suivantes 7.4 et 7.5 étudient le rôle de l'idée de *machine* dans l'élaboration de la conception

2. Voir par exemple(Davis 2012).

contemporaine du savoir scientifique, dont la mécanique fournit le modèle. Nous suggérons que les deux interprétations principales de son rôle, la première qu'elle permit de montrer la possibilité de mathématiser le mouvement, la seconde qu'elle mit en avant les « explications structurelles » des phénomènes de la nature, s'appuient l'une et l'autre sur un fait qui revêtait une importance capitale pour les modernes, celle de la *constructibilité* géométrique des machines, comme modalité essentielle de la résolution rigoureuse des problèmes. Nous avançons enfin que les notions de *machine* et de *constructibilité*, quoiqu'elles passent à l'arrière-plan avec l'avènement de la science newtonienne, restent toujours actives dans l'idée que nous nous faisons de la résolution scientifique des problèmes.

Enfin, en section 7.6, nous étudions les tribulations du terme *système* à l'époque moderne et son remplacement par *théorie* dans le dernier tiers du XVIII^e^ siècle pour désigner les doctrines scientifiques, du fait de son association malencontreuse avec la notion d'*hypothèse* et avec les querelles métaphysiques du Grand Siècle.

7.1 L'évolution du couple théorie / pratique jusqu'à la Renaissance

Cette section commence par exposer les transformations importantes de la notion de connaissance au Moyen Âge, dans un cadre encore aristotélicien : l'association de la connaissance spéculative à la théologie, la reconnaissance, par diverses voies, de la complémentarité, voire de l'unité de la théorie et de la pratique des choses du monde, enfin la caractérisation de la connaissance pratique comme *ordinatio* ou connaissance de l'ordre des choses – tous ces mouvements préparent la « révolution pragmatique » des Temps modernes (§ 111). Nous décrivons en particulier le grand mouvement de « rationalisation des pratiques » que furent les réductions en art (§ 112), et l'avènement du concept de *méthode*, promu notamment par La Ramée, qui vient prendre le relais de celui d'*ordinatio* (§ 113).

§ 111. L'unité spéculative de la connaissance selon Thomas

Les premières transmissions de la distinction entre théorie et pratique dans la tradition latine sont faites par Augustin et Boèce, qui la chargent de connotations chrétiennes. Augustin, lorsqu'il traduit *θεωρία* par *contemplatio*, s'intéresse sur-

tout au couple vie active / vie contemplative d'*Éth. Nic.* X. En rapprochant ce texte du récit évangélique de Marthe et Marie, il charge la *contemplatio* d'un sens religieux qui ne connote plus qu'accessoirement l'étude de la physique et des mathématiques. L'influence néo-platonicienne d'Augustin oriente également le terme vers la désignation de l'acte parfait et achevé de la vie théorique, c'est-à-dire la vision directe des vérités divines.

Boèce, voulant plutôt traduire *θεωρία* dans le cadre de la distinction entre philosophie théorique et philosophie pratique, ne se satisfait pas de ces connotations trop étroites de la *contemplatio* et choisit à la place *speculatio*[3]. Le terme, qui signifie initialement *explorer, guetter*, est utilisé par Cicéron pour désigner les physiciens qui « observent et explorent la nature[4] », mais également par Jérôme pour traduire aussi bien la vision d'un point haut, au sens propre, que la vision prophétique ; par ailleurs *speculatio* garde la connotation de l'examen et de l'investigation ; il apparaît donc plus à même de traduire la richesse de sens de la *θεωρία* aristotélicienne, et sera prédominant au Moyen Âge latin. La *speculatio* reste néanmoins résolument orientée vers la recherche de Dieu. Un jeu de mots sur le célèbre verset de Paul (1 Cor 13,12) : « À présent nous voyons comme à travers un miroir (*speculum*), mais alors nous verrons face à face », permet aux théologiens latins de développer l'idée que la spéculation sur les choses naturelles a pour finalité profonde de contempler Dieu à travers le miroir qu'est la Nature. Mais cela implique que l'idée aristotélicienne, selon laquelle la vie théorique s'occupe de la connaissance « pour elle-même », n'a plus vraiment d'autre objet que Dieu lui-même ; les mathématiques et la physique perdent, à strictement parler, leur statut de connaissance « désintéressée ».

De ce fait, la complémentarité, voire l'unité entre la pratique et la connaissance théorique des choses du monde, va commencer à être aperçue, de plusieurs perspectives différentes. Nous en identifions trois ici.

D'abord, la tradition arabe livre au Moyen Âge latin l'idée qu'on peut distinguer, *à l'intérieur d'une même science*, une partie théorique et une partie pratique. Cet usage apparaît d'abord pour la médecine, peut-être dans la tradition médicale alexandrine du VI^e^ siècle[5]. Il est attesté chez les médecins arabes Hunayn Ibn Ishaq (801 - 873), Ali Ibn Abbas (930 - 994), Avicenne (980 - 1037) et se propage par le

3. (Kerstiens 1958, p. 383).
4. *De Natura Deorum, 1, 83*.
5. Voir Cunningham (1982, p. 407-413).

biais de leurs traductions dans le monde latin. Mais c'est dans le traité d'Al-Farabi (872 - 950) sur la *Division des Sciences*, que cet usage est appliqué de manière large à plusieurs disciplines, et notamment aux disciplines mathématiques, explicitant ainsi une division socio-culturelle ancrée dans l'Antiquité[6]. Ainsi, l'arithmétique théorique, qui étudie les nombres « pris absolument et en tant qu'ils sont abstraits par l'esprit des corps et de tout ce qui est nombrable » est distinguée de l'arithmétique pratique « qui étudie les nombres [des choses que l'on compte] […] comme *des hommes, des chevaux, des dinars, des dirhams*, et tout ce qui peut être compté et qui est l'objet de transactions publiques dans les marchés et la vie civile[7] », par quoi Al-Farabi entend sans doute les différents systèmes marchands de mesure ainsi que leurs règles de calcul et de conversion, qui occupent une bonne partie des manuels d'arithmétique au Moyen Âge. De même, la géométrie théorique est distinguée de la géométrie pratique « qui étudie les lignes et les plans dans un corps » que celui-ci soit le bois du menuisier, le fer du forgeron, le mur du maçon, ou le champ d'un arpenteur[8]. Il est intéressant pour notre propos de noter qu'Al-Farabi définit la mécanique comme la discipline chargée de la *réalisation* des conceptions mathématiques dans le monde matériel, et l'algèbre (par quoi il entend ce que nous appellerions l'algorithmique) comme une « mécanique arithmétique » qui donne les méthodes pour « déduire les nombres »[9]. Cependant cette idée ne semble pas avoir eu de postérité notable.

Le traité d'Al-Farabi va être traduit en latin au XII^e^ siècle. Au même moment, Hugues de Saint-Victor (? - 1141) propose une quadri-partition du savoir en *théorique, pratique, mécanique et logique*[10]. La *théorique* – Hugues reprend le terme grec, mais uniquement sous la forme adjectivale *theorica* – comprend la théologie, les mathématiques du quadrivium, et la physique. Par *pratique* il vise (comme Kant le fera plus tard) le domaine éthique, et c'est sous le terme générique de *mécaniques* qu'il rassemble les techniques comme l'habillement, la navigation, la construction, l'agriculture, etc. *Mécanique* ne se réfère ici que lointainement à l'usage des machines, et Hugues l'associe aux adjectifs *servile* et *adultérin* (d'après un jeu de mot sur *mechus*, adultère, car la technique créée des artifices par l'union de l'intelligence et de la nature) – termes péjoratifs qui constitueront l'essentiel de son sens

6. (Asper 2009).
7. *Le recensement des sciences*, éd. Cherni, p. 125.
8. ibid., p. 130.
9. ibid., p. 160.
10. *Didascalion de studio legendi*, livre II, chapitre 1, cité par (Châtillon 1965).

jusqu'aux Temps modernes[11]. Cependant Hugues a le mérite de reconnaître les techniques comme champ de savoir authentique[12] et d'élargir ainsi sans précédent l'horizon de la connaissance, jusque là réduit à l'enseignement de l'Université. La classification d'Hugues et celle d'Al-Farabi vont se conjoindre et former la base de la vision scolastique des techniques.

Par Al-Farabi, l'idée devient donc courante dès le XII^e^ siècle que les sciences théoriques peuvent être d'une grande utilité pour les arts mécaniques[13]. Thomas d'Aquin constate ainsi que certains savoirs sont purement spéculatifs (ainsi les mathématiques), d'autres purement pratiques (ainsi l'agriculture) et d'autres « spéculatifs sous un aspect, et pratiques sous un autre »[14], comme la médecine. Cette nouvelle utilisation de l'opposition *théorie / pratique*, qui peut nous paraître naturelle, brouille significativement la clarté du schéma aristotélicien. Elle remet en cause son fondement ontologique, à savoir la distinction entre objets nécessaires et contingents. Thomas explique cela en distinguant trois sens possibles par lesquels on peut qualifier une connaissance de *spéculative* ou de *pratique*, selon son objet, selon sa modalité, et selon sa finalité. Cette distinction lui permet d'étendre très loin le domaine de la connaissance spéculative[15] :

1. Une connaissance est purement spéculative lorsqu'elle se rapporte à un objet qui ne peut être produit par le sujet connaissant : ainsi les objets naturels, sur lesquels nous ne pouvons que spéculer.
2. On peut également observer un artefact de manière spéculative, ainsi une maison lorsqu'on « réfléchit à son genre, ses différences et ses propriétés » de telle manière à la définir ou à la classer, en suivant une démarche analytique.
3. Un architecte peut aussi réfléchir à la manière de construire cette maison, mais sans en avoir l'intention, se demander « comment elle pourrait être construite, non afin de la construire, mais simplement pour le savoir ». La connaissance est ici « pratique virtuellement », mais elle est spéculative « selon la fin », car c'est seulement la connaissance qui est l'objectif ici de la réflexion[16].

11. Sur le sens de *mécanique* au début des Temps modernes, voir (Gabbey 2004, p. 12).
12. (Châtillon 1965, p. 546).
13. (Beaujouan 1957, p. 10).
14. *Somme Théologique* I q. 14, a. 16.
15. Nous suivons ici de près les deux textes parallèles du *de Veritate*, q. 3, a. 3. et de la *Somme Théologique* I q. 14, a. 16.
16. *de Veritate*, q. 3, a. 3.

Pour Thomas, – et il s'agit là du deuxième déplacement majeur opéré sur le couple théorie / pratique, aucune de ces distinctions n'est absolue ; seule la dernière distinction est pertinente :

> Quand nous distinguons la philosophie ou les arts en théorique ou pratique on doit le faire sur la base de leur finalité, en nommant théorique ce qui est seulement dirigé vers la connaissance de la vérité, et pratique ce qui est dirigé vers l'opération[17].

Thomas considère ainsi qu'on peut avoir une connaissance spéculative des artefacts et des méthodes de l'action, mais ce n'est pas tout : dans un coup de force décisif, Thomas achève de démontrer l'unité de la connaissance en remarquant que la distinction de la connaissance selon sa finalité, la seule qu'il juge pertinente, lui survient de l'extérieur. Elle ne lui est pas même essentielle :

> Un élément accidentel dans l'objet qui spécifie une puissance ne la diversifie pas, nous l'avons déjà dit. Il est accidentel à l'objet coloré qu'il soit homme, qu'il soit grand et petit ; aussi tout cela est-il saisi par la même puissance de voir. Or il est accidentel à un objet saisi par l'intelligence qu'il soit ordonné à l'action ou non. Et c'est en cela que diffèrent intellect spéculatif et intellect pratique. L'intellect spéculatif est celui qui, lorsqu'il appréhende quelque chose, ne l'ordonne pas à l'action, mais seulement à la contemplation de la vérité. Au contraire l'intellect pratique ordonne à l'action ce qu'il appréhende[18].

La position de Thomas est que l'intellect pratique « étend » l'intellect spéculatif par la considération de la fin, en s'appuant sur le texte de *De anima* III.7. Il se situe en cela dans la tradition de Jean Philopon (VI^e^ siècle), selon qui l'intellect deviendrait pratique en ajoutant, à la considération du vrai et du faux, celle du bon et du mauvais. *Bon et mauvais* doivent être entendus ici en leur sens le plus général, qui comprend aussi bien leur valence absolue (morale) que relative (utilitaire), et c'est cette dernière qui nous intéresse ici. Cette interprétation présente le savoir pratique comme une *modalité* qui se surajoute au savoir simple, essentiellement spéculatif[19].

Un troisième déplacement est enfin opéré si on considère l'intellect divin : Dieu ne connaît pas en effet les choses en tant qu'il les appréhende, comme si elles étaient extérieures à lui, mais en tant qu'il en est le créateur et la cause. Son intelligence des choses est donc pratique :

> Dieu connaît les choses créées à la manière dont l'artisan connaît les produits de

17. *Super Boetium De Trinitate* q. 5, a. 1.
18. *Somme Théologique* I q. 79, a. 11.
19. Jean Philopon, *Commentaire du Traité de l'Âme*, CAG XV, p. 554-555.

> l'art, laquelle connaissance est la cause des produits de l'art. Le rapport de cette connaissance aux choses connues est donc l'inverse de celui qu'entretient notre connaissance[20].

Cette idée, qui remonte à Augustin[21] peut être aisément rattachée à la tradition médiévale du *Timée*, et la connaissance technique dont parle Thomas porte bien sûr sur des causes finales – on est donc encore loin de l'idée d'un « Dieu mécanicien » qui apparaît à la fin du XVI^e^ siècle. Elle permet néanmoins l'émergence d'une posture épistémologique qui traverse les Temps modernes, de Nicolas de Cuse à Descartes, et que l'historien des sciences A. C. Crombie appelle *modélisation hypothétique* : à savoir que connaître une chose, c'est connaître comment on pourrait la faire[22].

Quelle est donc la structure de cette connaissance pratique ? Sa principale caractéristique selon Thomas réside dans la notion d'*ordinatio*, que nous traduirons ici, dans le contexte précis qui nous intéresse, par *ordonnancement*. Il s'agit d'abord d'un concept théologique, qui décrit l'orientation fondamentale de toute la création vers Dieu, et qui sous-tend la conception thomiste de la providence divine[23]. Mais il a également un intérêt épistémologique, car l'*ordonnancement* apparaît sous trois aspects : il y a d'abord celui de la finalité (l'ordre de l'usage), puis celui de la structure (l'ordre des parties entre elles) et enfin celui de la procédure (l'ordre de la génération) – eux-mêmes étant ordonnés les uns aux autres comme par emboîtement :

> Il faut donc savoir que l'artiste ou l'artisan, en concevant à l'avance son œuvre, considère premièrement la fin ; et ensuite il considère l'ordre de la chose qu'il cherche à faire par rapport à cette fin, et aussi l'ordre des parties entre elles, par exemple que les fondations sont sous les murs et les murs sous le toit ; et cet ordre des parties entre elles est finalement ordonné à la fin de la maison. Troisièmement, il faut qu'il considère les choses par lesquelles il progresse dans la poursuite de la fin et qu'il fasse disparaître celles qui empêchent de la réaliser […] En raison de l'ordre de ce qui est imaginé par rapport à la chose qui doit être faite, elle est appelée du nom de disposition : car la disposition signifie un certain ordre ; d'où l'on dit que la disposition est une mise en ordre de la génération[24].

20. *de Veritate*, q. 2, a. 8.
21. (Crombie 1994, vol. 2, p. 1088).
22. (ibid., vol. 2, p. 1081-1241).
23. (Durand 2012).
24. *1 Sent. d. 39, q. 2, a. 1.*

Ces schémas épistémiques sont très différents de ceux qu'enseigne la logique classique, centrée autour des catégories, des prédicables et des universaux. Duns Scot reprend la même leçon que Thomas, et utilise le concept d'*ordonnancement* pour expliquer la différence que fait *Mét.* A.1 entre expérience et technique :

> Il y a deux sortes de savoirs pratiques : savoir agir (*scire operari*), et savoir ce qui est ordonné à l'agir (*scire ordinatum ad operari*). Le premier est proprement pratique ; le second est appelé pratique au sens large. Et ainsi, tout homme de l'art a un savoir pratique au second sens, mais non au premier[25].

Cette distinction permet à Duns Scot d'expliquer en quelle mesure la connaissance pratique n'est pas réduite à la connaissance des cas particuliers : si tel est le cas pour le savoir-faire, assimilé à la simple habileté de l'homme d'expérience, « la connaissance de l'art est par elle-même universelle, et si elle est connaissance du singulier, c'est à titre de conséquence et par accident » – remarque qui rejoint notre propre analyse de la *technè* aristotélicienne comme universalisation d'une délibération. L'*ordonnancement*, qui est l'acte de mettre en ordre *en vue d'une fin* joue ici un rôle central dans cette nouvelle modalité d'accès à l'universel : elle donne à la connaissance pour objet des structures fin / moyens dont on peut voir l'identité à travers une diversité de situations.

Dans le cadre de notre propos, nous ne pouvons manquer de souligner que notre *ordinateur* dérive directement d'*ordinatio* pris en ce sens. Depecker (2015) raconte en effet que ce terme fut proposé en 1955 par Jacques Pellet, professeur de latin médiéval à la Sorbonne, à IBM France « qui cherchait un équivalent français au terme *computer*, de façon à montrer sa bonne intégration en France à une époque d'anti-américanisme latent ».

§ 112. La réduction en art

Affaiblissement de l'idéal de connaissance désintéressée (hors la recherche de Dieu), caractère accidentel de la distinction entre connaissance théorique et connaissance pratique, primat dans l'intellect divin du modèle pratique du savoir : l'essentiel des déplacements conceptuels concernant le couple théorie / pratique est déjà accompli *au sein même* de la scrupuleuse réception latine d'Aristote. Les Temps modernes, si critiques en général envers le Stagirite et l'École, vont sur ce sujet se contenter de résoudre les tensions opérées par ces déplacements.

25. *Questions sur la Métaphysique*, éd. Boulnois, p. 278.

D'abord, l'association des activités théoriques à l'activité religieuse fragilise leur position privilégiée parmi les vocations humaines. Il y a toujours eu, au sein de l'Église et de la réflexion théologique elle-même, un courant de méfiance pour l'attitude intellectuelle pure, entachée du risque de « curiosité ». Au XII^e^ siècle déjà, Jean de Salisbury affirme que la spéculation doit n'être recherchée que lorsqu'elle porte des fruits – ce qui contredit expressément l'idéal aristotélicien[26]. Au XIV^e^ siècle, Jean Gerson appelle au retour à la foi pure, débarrassée des constructions intellectuelles concernant Dieu, préfigurant ainsi le rejet humaniste de la spéculation scolastique[27]. La plus virulente critique religieuse de l'activité spéculative vient cependant de Luther qui rejette l'idée que la raison naturelle puisse s'élever à la compréhension de Dieu. Nous ne pouvons connaître celui-ci que par sa propre révélation personnelle, comme Dieu qui agit dans l'histoire[28]. À travers ce courant émerge l'idée que l'accès à Dieu ne passe pas par la réflexion spéculative, contrairement à ce qu'affirmait Thomas.

Parallèlement à cette critique religieuse des activités spéculatives, l'humanisme revalorise la vie active, et l'idéal de la vie bonne. Là encore, on peut trouver des antécédents à ce courant jusqu'au XII^e^ siècle au moins, chez Bernard de Clairvaux qui valorise la *praxis*, le travail organisé et discipliné des moines comme chemin privilégié d'accès à Dieu[29]. Au XIV^e^ siècle, Collucio Salutati affirme que l'objectif des êtres humains n'est pas de connaître Dieu par la connaissance spéculative. Se référant à Cicéron, il argumente que la vie active est le but de la personne, inaugurant ainsi une tradition de « pragmatisme humaniste[30] ».

Cette évolution des valeurs traduit en partie une évolution sociétale : des artisans établis et éduqués visent à la reconnaissance de la noblesse de leurs arts et s'insurgent contre leur qualification d'*arts serviles*. Ils cherchent à montrer qu'ils requièrent ingéniosité et réflexion tout autant que les arts libéraux.

Cette valorisation du savoir pratique devient un étendard des philosophes modernes contre les « anciens ». Le commandement fait à l'étudiant d'aller visiter les ateliers des artisans devient un *topos* récurrent des modernes, de La Ramée jusqu'à Leibniz, en passant par Rabelais, Descartes, etc.

Ce sont à présent les arts, tant libéraux que mécaniques, qui sont au centre

26. (Kerstiens 1958, p. 389).
27. (Ebbersmeyer 1995).
28. Voir la mise au point d'Oberman (2003) concernant la relation de Luther à la philosophie.
29. (Stock 1975).
30. (Ebbersmeyer 1995).

des enjeux de la connaissance, et la Renaissance voit la diffusion « en nombre presqu'incroyable »[31] – grâce également à l'imprimerie – de manuels leur étant consacrés, selon le modèle de la *réduction en art*.

Les *réductions en art* sont des entreprises de description raisonnée et méthodique des savoir-faire pratiques d'une discipline donnée. Hélène Vérin a fait l'histoire de ce grand mouvement intellectuel[32], central pour notre propos, qui traverse les Temps modernes. L'idée aussi bien que l'expression *réduire en art* (*ad artem redigere*) proviennent de l'Antiquité latine. Cicéron expose ce besoin de rationalisation à propos du droit, mais c'est le traité sur l'architecture de Vitruve qui en est le modèle pour la postérité. Le genre prend une très grande ampleur avec l'invention de l'imprimerie[33]. Tous les arts sont concernés, libéraux aussi bien que mécaniques : agriculture, mines, guerre, fortifications, construction navale, rhétorique, danse, escrime, etc. mais également par extension des disciplines éthiques comme l'éthique et la politique. Hélène Vérin décrit ainsi cette activité de « rationalisation des savoirs pratiques[34] » :

> Réduire en art est rassembler, mettre en ordre et diffuser par l'écrit ; rassembler des données dispersées, confuses voire profuses, les ordonner, les éclaircir à l'aide de règles et de préceptes, dans un exposé bref et méthodique, et diffuser par écrit le résultat de ce travail dans un langage accessible à tous, afin que chacun dispose des moyens d'agir au mieux, ses choix étant ainsi facilités, conduits, réglés. Par ce travail de réduction, l'auteur contribue à la diffusion d'un savoir et donc à son perfectionnement possible, grâce au travail successif – et cumulatif — des générations ; la finalité toujours évoquée est le bien public. Le point de vue est celui de l'action : sur les idées, sur les hommes, sur les choses[35].

On se trouve donc précisément dans ce que Thomas d'Aquin appelait la spéculation qui prend pour objet la pratique, à une époque où cette opposition est en rapide évolution :

> Le *De Re Metallica*[36] prit racine la pensée humaniste, entre université, administration et médecine, et plus concrètement encore, dans le réseau des familiers d'Érasme. Le souci majeur de ces intellectuels du XVI^e^ siècle se situait, en amont d'une quel-

31. (Gilbert 1960, p. 69).
32. Les deux ouvrages de référence sur cette question sont (Vérin 1993, 2008).
33. (Vérin 2008, p. 13, 16).
34. (Vérin et Dubourg Glatigny 2008).
35. (ibid., p. 19).
36. Un des ouvrages emblématiques du XVI^e^ siècle,de Georgius Agricola (1494 - 1555), qui entreprend d'exposer l'art des mines.

> conque distinction entre science et technique, dans la nécessité qu'ils ressentaient de faire évoluer la pensée théorique, de l'amener à se préoccuper de pratique, de lui donner les moyens d'investir la réalité[37].

La fonction de ces ouvrages est multiple. Outre la sauvegarde et la diffusion des savoirs pratiques, ils visent à éduquer les administrateurs, les juristes et les commanditaires de projets toujours plus vastes et onéreux, qui doivent donc pouvoir comprendre les normes des métiers avec lesquels ils ont affaire[38]. Il s'agit, globalement, d'augmenter le contrôle de l'action et l'assurance de son succès, son *effectivité* ou encore, comme le disent Dubourg Glatigny et Vérin (2008, p. 50) l'émergence d'« une rationalité partagée, centrée sur l'efficacité et le respect des règles ». Enfin, en exposant l'état actuel des arts, ces ouvrages espèrent susciter l'invention, en particulier celle qui pourrait provenir des savants, principalement les géomètres, capables d'examiner les raisons de la matière assemblée et de les corriger[39].

D'un point de vue méthodique, les réductions en art s'inspirent des sources antiques lorsqu'elles existent, mais aussi et surtout de la pratique et des inventions contemporaines. Il s'agit donc de visiter les ateliers et d'interroger les artisans, voire de pratiquer l'art soi-même, bien que ce ne soit pas le cas général. En effet, la réduction en art requiert une compétence spécifique, celle de la dialectique[40] qui est capable de voir l'universel dans le particulier, de faire dire à l'ouvrier ce qui pour lui va sans dire, de décrire des opérations délicates qui se transmettent par le geste et non par la parole. La réduction en art passe donc par un travail très important de constitution et de normalisation des vocabulaires techniques[41]. La réduction en art est également un jalon important concernant la transmission visuelle et diagrammatique des savoirs, représentant les décisions par des arbres dichotomiques, par des tables quantitatives, et par des gravures représentant les machines, les ateliers et les gestes de l'ouvrier ; dans le cas des arts performatifs comme la danse et l'escrime, c'est tout un langage, à la fois visuel et textuel qui est inventé[42].

37. (Garçon 2008, p. 269).
38. (Vérin et Dubourg Glatigny 2008, p. 25, 35), (Dubourg Glatigny et Vérin 2008, p. 56).
39. (Vérin et Dubourg Glatigny 2008, p. 36), (Dubourg Glatigny et Vérin 2008, p. 56).
40. (Vérin et Dubourg Glatigny 2008, p. 21).
41. (Dubourg Glatigny et Vérin 2008, p. 59).
42. (ibid., p. 60).

§ 113. La Ramée - méthode et système

Les réductions en art ne sont pas perçues comme des entreprises de spécialisation des savoirs. Leur possibilité même montre au contraire qu'il n'est pas de savoir qui ne soit accessible à quiconque sait bien user de sa raison. Elles servent et inspirent ainsi la cause des humanistes, celle d'un idéal unitaire et non disciplinaire de la connaissance. Il s'agit avant tout de développer le *penser par soi-même*, et les arts libéraux sont perçus comme les lieux idoines de l'exercice de l'esprit menant à son plein développement. Une des raisons de la revalorisation des arts au XVI^e^ siècle, outre leur utilité pour la vie bonne, est qu'ils présentent le modèle d'une connaissance *active*, qui requiert invention, jugement et méthode.

Il s'agit là des trois parties de la logique nouvelle que promeut Pierre de La Ramée (1515 - 1572) dans sa *Dialectique*, plusieurs fois réécrite tout au long de sa vie, de 1543 jusqu'à sa mort en 1572. Les préoccupations de La Ramée ne sont pas dirigées vers l'investigation de nouveaux savoirs ; elles concernent plutôt l'apprentissage et la transmission de savoirs qu'il semble juger à peu près constitués. Cela implique que la principale mission du savant n'est pas de chercher, mais de « bien disputer », et La Ramée affirme parfois que la rhétorique a la prééminence parmi les arts du *Trivium*. Cependant, il faut entendre « disputer » dans la tradition platonicienne qui définit la pensée comme un dialogue avec soi-même, soit comme la discursivité en général et « ainsi l'art de connaître, c'est-à-dire dialectique ou logique, est une et même doctrine pour apercevoir toutes choses[43] ».

La tripartition de la dialectique suit l'ordre logique de la constitution de la connaissance : les arguments sont trouvés par l'invention à l'aide des lieux communs, le jugement simple (c'est ainsi que La Ramée appelle le syllogisme) permet de vérifier leur validité, en les inférant d'arguments déjà connus, et enfin la méthode organise l'ensemble des arguments en un tout homogène, cohérent et ordonné à partir des axiomes.

L'emploi du terme *méthode* par La Ramée n'est pas anodin. Lorsque le terme *methodus*, d'origine grecque, commence à se répandre chez les humanistes, il signifie pour eux « une forme abrégée d'art » c'est-à-dire justement les réductions en art, dont quelques-unes, dès le début du XVI^e^ siècle, portent déjà le nom de *Méthode*[44]. La Ramée n'a donc pas inventé cette notion de toutes pièces : il pouvait

43. *Dialectique* (1555), éd. Bruyère, p. 121. Ce passage est cité et commenté par (Bruyère 1984, p. 223).

44. (Gilbert 1960, p. 69). Voir l'annexe, p. 233 pour une liste de tels ouvrages, que Gilbert

en observer le phénomène de manière très concrète à son époque[45].

Avec la méthode, c'est bien l'idée de *système* qui est déjà là. La Ramée connaît la définition stoïcienne de la *τέχνη* transmise par Lucien :

> Τέχνη ἐστίν [...] *σύστημα ἐκ καταλήψεων συγγεγυμνασμένων πρός τι τέλος εὔχρηστον τῷ βίῳ*[46].
>
> L'art [...] est un ensemble (*système*) de notions mises en pratiques pour quelque fin utile à la vie.

La Ramée écrit :

> *Ars est comprehensio praeceptorum in rebus aeternis, propriorum et ordine dispositorum, ad utilem vitae finem spectantuim*[47].
>
> L'art est la compréhension des préceptes dans les choses éternelles, propres et ordonnées, visant la fin utile de la vie.

La Ramée utilise *comprehensio* pour traduire le terme grec de la citation de Lucien, mais après cette citation il renvoie au livre II de sa *Dialectique* qui porte sur la méthode : le lien est donc explicite. De plus, Melanchthon, son contemporain et correspondant, traduit dès 1534 la définition de Lucien en utilisant la forme latinisée *systema*. Comme ce dernier fut très lu en pays protestants, on suppose qu'il s'agit là d'une des sources principales de diffusion du terme en ce sens, bien après lui cependant, vers les années 1570-1580[48]. Il est alors employé, dans les milieux protestants, pour désigner des ouvrages de didactique chrétienne, exposant de manière ordonnée les vérités de la religion, qui sont conçues, en parallèle de l'image de l'Église comme corps du Christ, comme le *corpus integrum* des vérités contenues dans le Nouveau Testament, dont aucune ne peut être ôtée sans que l'édifice complet ne s'écroule.

Étant donné l'arrière-plan ramiste qui rend le terme très proche de *méthode*, et l'arrière-plan plus profond encore des réductions en art, il n'est pas étonnant qu'une fois adopté, *systema* se sécularise et se généralise rapidement pour désigner l'exposition ordonnée d'un domaine de la connaissance. Les trois « systématisateurs » les plus connus sont à cet égard Bartholomeus Keckermann (1572 - 1609), Clemens Timpler (1563 - 1624) et Johann Heinrich Alsted (1588 - 1638), qui les

affirme seulement suggestive. Voir également (Ong 1958, p. 228).

45. Nous suivons là la suggestion de Vérin et Dubourg Glatigny (2008, p. 15).
46. Lucien, *Le parasite*, chp. 4, SVF 1.73.
47. Cité par (Schmidt-Biggemann 1983, p. 44, n. 162). Le texte original des *Praelectiones* de 1572 est difficilement accessible.
48. (Ritschl 1906, p. 10-24).

premiers publient des ouvrages portant le titre de *Système*. Ce titre promet au lecteur un exposé ordonné et didactique de la discipline en question. Keckermann, par exemple, est l'auteur de *Systèmes de Logique, de Rhétorique, de Politique*, etc. Mais le grand'œuvre de ce moment de l'histoire de la compilation des savoirs est l'*Encyclopédie* d'Alsted, publiée en sept livres en 1630, sur 2500 pages. Cet ouvrage ne vise pas seulement à rassembler le savoir essentiel de toutes les sciences, de sorte à rendre tous les autres livres inutiles, mais il se présente également, dans une lointaine inspiration lulliste, comme le complément même de la philosophie, un « cursus encyclopédique » qui doit éduquer les facultés de l'esprit – invention, jugement, mémoire[49].

Comme nous l'avons déjà dit, ces ouvrages que sont les méthodes et les systèmes, dont la variété, le nombre, le volume et la diffusion vont croissant aux Temps modernes, ne sont pas vus comme une spécialisation et une fragmentation des savoirs. Les méthodes particulières ne sont, pour La Ramée, que des instances d'une même méthode qui peut en droit s'occuper de tous les sujets de connaissance ; elles ne font que la guider et soutenir dans ces entreprises.

En effet[50], la dialectique « artificielle » que La Ramée expose et enseigne n'est que l'image d'une dialectique naturelle que tout être humain possède : la première n'est là que pour développer et guider la seconde lorsqu'elle est pervertie. La dialectique est, mieux qu'une faculté, une *virtus*, terme dans lequel il faut entendre la *vis* originelle de l'esprit actif et naturellement disposé à connaître, c'est-à-dire une *force* qu'il faut exercer afin qu'elle devienne habitude et connaissance effective. Il suit de là que la méthode est également l'image d'une méthode « naturelle », celle qui enjoint l'esprit de rassembler son savoir en un tout cohérent et de l'ordonner à partir de ce qu'elle y trouve de plus clair, qui est également ce qu'il y a de plus général et premier, dont tout le reste procède. La méthode est donc unique et universelle, car elle est le principe même qui rend possible les méthodes multiples que sont les réductions en art :

> Cette méthode est singulière et unique ès doctrines bien instituées car en elle, singulière et unique, est procédé par choses antécédentes du tout et absolument plus claires et notoires pour éclaircir et illustrer les choses conséquentes et inconnues[51].
>
> C'est pourquoi cette méthode doit être admirée non seulement pour sa clarté, mais aussi pour sa concision […] Par une telle voie, il sera permis de connaître plus

49. (Schmidt-Biggemann 1983, p. 131-139).
50. Dans ces paragraphes, nous nous appuyons sur (Bruyère 1986).
51. *Dialectique* (1555), éd. Bruyère, p. 121, cité et commenté par (Bruyère 1984, p. 136).

rapidement et de manière plus concise[52].

Cette dialectique entre méthode naturelle et méthode artificielle se retrouve également, sous d'autres termes, chez les systématisateurs allemands. Par exemple, pour Keckermann, *Logique* doit s'entendre d'une double manière. Elle est d'abord une disposition naturelle que l'habitude acquise par les préceptes et l'exercice doit développer. Elle est par ailleurs le *système* explicite de ces préceptes, c'est-à-dire leur exposé didactique *extérieur*. Le système joue donc chez Keckermann le même rôle que la méthode artificielle chez La Ramée : il est l'extériorisation discursive, dans les préceptes de la technique, du savoir-faire intérieur et véritable d'une pratique donnée[53].

7.2 La recherche de la systématicité

Peut-on dire que ces réductions en art et ces idées nouvelles de *méthode* et de *système* forment une rupture relativement à la conception aristotélicienne de la technique, et qu'elles annoncent cette « technique systématique » dont nous avons esquissé la possibilité à la fin du chapitre précédent ?

La question est difficile, car il semble bien y avoir une ambiguïté réelle des intentions des réductions en art, qui est visible dans l'opposition qui est faite à l'époque, et qui peut être surprenante pour nous, entre les idées de *routine* et de *règle* (§ 114). La solution que nous adopterions aujourd'hui et qui, en gros, dirait que les règles techniques visent à mettre en place des routines rationnelles, est entrevue par Leibniz mais il considère cette possibilité comme un pis-aller de la connaissance, auquel il oppose son projet de Caractéristique Universelle (§ 115). Dans le dernier développement § 116, nous esquissons la solution qui fut finalement trouvée à cette tension, celle du *découplage* entre la conception de la solution d'un problème, que l'art peut seulement guider, et son exécution, que l'art doit rendre infaillible, voire automatique. Nous développerons cette thématique à la section suivante.

52. *Usus artificosae methodi* (1554), cité et commenté par (Bruyère 1984, p. 137).
53. (Ritschl 1906, p. 27-29).

§ 114. L'ambiguïté des règles

Les adversaires de La Ramée ne manquèrent pas de souligner la proximité de sa méthode avec la doctrine aristotélicienne de la science examinée au chapitre précédent[54]. La Ramée, qui fit carrière de son anti-aristotélisme virulent, ne le reconnut que tardivement, et contr'attaqua en reprochant à Aristote de n'avoir jamais appliqué sa doctrine à ses propres ouvrages, en ne les mettant pas en système[55].

Il y a cependant bien une différence cruciale : tandis qu'Aristote l'avait réservée au savoir scientifique, la méthode porte sur tout domaine de connaissance – et en particulier sur les domaines contingents de l'action. Exhiber la forme systématique d'arts qui ne disposent pas de principes premiers (au moins évidents) est un problème nouveau. Comment guider la réflexion par ordre, du plus simple au plus composé, de manière à ne rien oublier ?

Au niveau le plus élémentaire et le plus pragmatique, il s'agit d'aider le lecteur à *trouver* l'information pertinente dans le savoir mis à sa disposition. Les ouvrages de la Renaissance vont prendre parti des nouvelles possibilités de l'imprimerie pour inventer les outils typographiques modernes – la table des matières, l'index thématique, de lieux, des personnes, etc.[56]

Parmi ces artifices visuels, les tables et les arborescences jouent un rôle particulier, car elles ont l'ambition de cartographier l'information selon leur logique interne. En particulier, les arborescences peuvent se réclamer de la diérèse platonicienne, qui développe les concepts dans toute la richesse de leurs significations. La Ramée fit notamment un usage si intensif des arbres dichotomiques qu'il en devint proverbial[57]. Cependant, les défauts de cette méthode, qu'Aristote avait relevés[58], sont ici patents : La Ramée changera sans cesse de principes de division à travers ces ouvrages, si bien qu'une impression d'arbitraire s'en dégage, dont se moquèrent ses adversaires[59].

Il y a cependant un autre usage des arborescences qui se fait jour lorsqu'il s'agit de classer des *règles* et des *préceptes* pour la décision[60]. Suivre un diagramme pour trouver quelle est la bonne décision à prendre face à un problème donné, c'est se

54. Voir plus haut, section 6.4.
55. (Bruyère 1984, p. 126-135).
56. (Blair 2010, ch. 3).
57. (Ong 1958, p. 199).
58. Voir plus haut, § 107.
59. (Ong 1958, p. 200).
60. (Vérin et Dubourg Glatigny 2008, p. 35).

laisser guider par un *questionnaire* qui demande quelles sont les caractéristiques décisives de la situation, c'est *réfléchir* à elle d'une manière nouvelle, ordonnée.

> Toute réduction en art doit définir et ordonnancer ses opérations. Mais encore, définitions et ordonnancement sont interdépendants. La voie à suivre dépend de la teneur des opérations précédemment retenues, plus exactement, de la puissance ou potentialité qui caractérise chacune d'elles. Cette rationalisation des choix successifs est fréquemment présentée comme un emboîtement d'opérations qui vont du général au particulier, ce qu'expriment les tableaux arborescents. Ils offrent un système ordonné de classes d'opérations de plus en plus déterminées. Les tables de Simon Stevin et de Jean Errard pour la fortification, celles de Testelin pour la peinture en sont des illustrations. Elles sont présentes dans tous les domaines où le moment de la conception est déterminant. L'objectif n'est pas de supprimer l'exercice du jugement mais de le guider, de l'abréger, de le faciliter[61].

Cependant, il peut arriver qu'à force d'abréviation du jugement, celui-ci soit devienne inutile. L'exemple des tables de Testelin, cité par les auteurs, illustre bien cette possibilité. Publiées en 1680, ces six tables visent à donner les règles de la peinture, dans le contexte de l'Académie Royale créée par Colbert[62]. Présentées chacune sous forme arborescente, elles énoncent les règles du trait, de l'expression, des proportions, du clair et de l'obscur, de l'ordonnance du sujet et de la couleur. D'après Christian Michel, qui les a étudiées en détail, elles contraignent de manière variable le choix du peintre : si celles de la proportion des figures humaines selon l'âge, la race, le sexe, laissent explicitement une assez large marge de manœuvre, « il en serait autrement de certaines parties d'autres tables, notamment celles où la recherche de règles, ou en tout cas d'un vocabulaire consensuel pour définir les questions qui se posent au peintre qui veut travailler par principes, se transforme en recettes[63] ».

Au sein même des réductions en art, il y a donc deux formes de règles.

Les premières, qu'on pourrait appeler *règles-guide*, sont semblables à celles que donne Aristote dans la *Rhétorique* : elles guident la réflexion et lui indiquent des points d'attention. Les tables et les arborescences peuvent organiser les questions auxquelles le jugement doit répondre, mais elles ne s'y substituent pas. La réduction en art ici ne vise pas à figer les conduites humaines dans la répétition de gestes serviles mais au contraire à les en libérer. De manière pour nous surprenante, la

61. (Dubourg Glatigny et Vérin 2008, p. 62).
62. (Michel 2008, p. 110).
63. (ibid., p. 115).

routine – terme attesté en français depuis le milieu du XVI^e^ siècle – est ce à quoi la règle justement s'oppose, comme l'écrit le *Dictionnaire de l'Académie* de 1694 :

> Routine. s. f. Usage acquis par une longue habitude, sans le secours de l'étude et des règles. *Il n'a jamais étudié à fond, mais il a acquis je ne sais quelle routine de discourir, de parler. Il fait cela par routine. il ne sait point de musique, mais il chante par routine.[…] Une vieille routine. La routine du monde, de la Cour*[64].

La routine désigne donc le savoir-faire qu'on acquiert par expérience et par coutume. Cela rend ce savoir particulier à un lieu et à un temps donné, et donc l'empêche d'être universel. La routine est *mécanique* au sens médiéval qu'avait ce terme, c'est-à-dire servile[65]. Un ouvrier, remarque Nicolas Aubin, auteur du premier *dictionnaire de marine* en 1702, est perdu lorsqu'on lui parle des pratiques de son métier qui ont cours dans la région voisine[66]. La routine n'a pas de règle, car il n'y a rien d'universel en elle.

Mais il y a également des *règles-contrainte* qui peuvent se présenter sous la forme de recettes. Les arborescences et des tables dans ce cas disent *exactement quoi faire*. L'arbre de décision est une ligne droite que trace la règle vers la résolution du problème, ligne que la diversité des circonstances ne parvient pas à casser, mais par laquelle au contraire elle se développe dans la succession de ses embranchements. La « rationalisation des choix successifs » que l'arbre opère conduit et abrège la réflexion car elle lui évite des détours inutiles et des cul-de-sac, c'est-à-dire ce que nous pourrions appeler des *faux problèmes*, qui sont autant de problèmes en moins à résoudre : des lieux où la réflexion aurait pu se perdre ou hésiter mais qui, une fois toutes les raisons correctement ordonnées, s'avèrent n'avoir qu'*une seule issue* et ne requérir, en fait, *aucun jugement*.

Cette dualité reflète la tension entre la disposition méthodique intérieure invoquée par La Ramée et le système extérieur qui en est le résultat tangible, couché en règles ordonnées dans des manuels. Car si la première manifeste l'autonomie de l'esprit en exercice, le second pourrait nous faire croire qu'il est à présent possible – grâce à son existence même – de s'en passer : il n'y aurait plus, en effet, qu'à appliquer les règles. Cette tension est aperçue par les modernes. Au XVII^e^ siècle apparaissent des satires qui se moquent de sots croyant maîtriser un art par la seule application des règles d'une méthode[67] ; et au XVIII^e^ siècle, se développe l'idée du

64. *Dictionnaire de l'Académie Française*, première édition, 1694, p. 417.
65. Voir plus loin § 120.
66. Cité par (Dubourg Glatigny et Vérin 2008, p. 59).
67. Voir par exemple, pour l'escrime (Brioist 2008, p. 248), et plus généralement, Molière.

génie qui, s'il maîtrise bien sûr les règles de l'art, sait également les transcender[68].

Cette tension est interne au concept de règle. D'une part – et c'est sans doute son sens prédominant de nos jours – la règle peut être comprise comme un universel dont le rapport au cas singulier n'est justement pas problématique, qui ne nécessite (idéalement) ni réflexion ni interprétation complémentaire pour pouvoir en être prédiqué – et ce rapport est justement ce que nous appelons *appliquer*. On dit de quelqu'un qu'il a *« appliqué la règle bêtement »* parce que la règle *permet* cette possibilité, parce que l'attitude la plus immédiate face à elle est de lui obéir *sans se poser de questions*. Faire preuve d'intelligence et de finesse requiert au contraire un certain recul par rapport à la règle, un certain art. Aussi y a-t-il également, d'autre part, un autre sens du terme, prédominant aux Temps modernes, qui est celui utilisé par Descartes lorsqu'il parle des *Règles pour la direction de l'esprit* dans une veine ramiste, et qui vise bien, au contraire d'une application aveugle, à éveiller l'attention aux considérations adéquates permettant la résolution de problème.

§ 115. Leibniz : la Caractéristique contre les procédures

Il est courant de voir en Leibniz celui qui, le premier, aurait pris au sérieux l'idée de réduire le raisonnement à un calcul, c'est-à-dire à l'application de règles aussi certaines et contraignantes que sont celles de l'arithmétique pratique. Ce serait donc par l'idée de calcul (qui serait aussi le socle de l'informatique) que celle de méthodes procédurales et systématiques de résolution de problèmes serait apparue. Cette idée expliquerait par ailleurs qu'ait été progressivement gommée la distinction entre règles et routines, puisque les règles des mathématiques pratiques peuvent le mieux être apparentées à des *routines*, sans pour autant qu'on pût les qualifier de telles, puisqu'elles exigent de l'étude.

Comme nous l'avons déjà dit, même si Leibniz a bien cette idée, et même s'il apporte à la notion de règle une inflexion capitale, il ne lui donne pas l'importance qu'on lui accorde aujourd'hui. Il nous faut à présent préciser ce point.

Le monde médiéval connaît de nombreuses règles de calcul, notamment pour déterminer le calendrier religieux, mais également des procédures pratiques, comme la multiplication et la division, les règles de conversion en écriture fractionnaire, les procédures pour extraire les racines carrées de nombres,

68. (Michel 2008, p. 107).

consignées par exemple dans le *Liber Abaci* de Fibonacci (1175-1250). Nous avons déjà cité le rapprochement intéressant qu'Al-Farabi fait entre mécanique et algorithmique[69].

Par ailleurs, l'idée d'une application de procédures de type arithmétique à la spéculation date au moins de Raymond Lulle, au XIII^e siècle[70], et l'idée d'un alphabet primitif de la connaissance, dont la manipulation par des procédures combinatoires générerait la vérité, est plus ancienne encore, se rattachant à la pensée théosophique et notamment kabbalistique. Ce courant profond de pensée irrigue la Renaissance jusqu'à Leibniz[71].

Enfin, dans une veine plus rationaliste, l'identification du syllogisme à une forme de calcul, par La Ramée, est reprise et étendue par Hobbes : « Computatio, sive Logica » est le titre de la partie logique du *De Corpore* (1655), et le célèbre chapitre cinq du *Léviathan* étend le domaine du calcul à toutes les disciplines de la raison. Cependant, chez La Ramée cette remarque doit être comprise en un sens dépréciatif, au service de son entreprise de démolition de la logique aristotélicienne et en particulier du syllogisme, puisque le calcul est une partie inférieure des mathématiques. Chez Hobbes, si l'idée a une tournure plus positive, et typiquement constructive (les opérations complexes de l'esprit doivent ainsi être comprises comme des compositions des opérations primitives que sont l'addition et la soustraction), elle ne se développe jamais en un projet d'étudier la logique de ces opérations.

C'est chez Leibniz qu'on trouve une première interprétation réflexive de toutes ces pistes. Le rapprochement de la pensée et du calcul n'est pas pour lui une analogie, ni une identité, mais un programme de recherches, qu'il exprime par l'objectif de transférer la certitude des procédures mathématiques à tous les domaines de la connaissance :

> J'ai remarqué que la cause qui fait que nous nous trompons si aisément hors des Mathématiques, et que les Géomètres ont été si heureux dans leurs raisonnements, n'est que parce que dans la Géométrie et autres parties des Mathématiques abstraites, on peut faire des expériences ou preuves continuelles, non seulement sur la conclusion, mais encore à tout moment, et à chaque pas qu'on fait sur les prémisses en réduisant le tout aux nombres ; […] L'unique moyen de redresser nos raisonnements est de les rendre aussi sensibles que le sont ceux des Mathématiciens, en sorte

69. Voir plus haut, § 111.
70. (Uckelman 2010, p. 429).
71. (Zweig 1997).

> qu'on puisse trouver son erreur à vue d'œil, et quand il y a des disputes entre les gens, on puisse dire seulement : comptons, sans autre cérémonie, pour voir lequel a raison[72].

Ce qui exprimé ici, c'est la certitude qui provient, non pas de l'intuition, comme chez Descartes, mais de la rigueur de la procédure elle-même. C'est le sens que Leibniz propose de donner au terme *méthode*, comme d'un fil d'Ariane, « sensible et grossier, qui conduise l'esprit comme sont les lignes tracées en géométrie, et la forme des opérations qu'on prescrit aux apprentifs en Arithmétique[73] ». De manière significative, lorsque Leibniz souhaite montrer comment cette rigueur de la réflexion mathématique peut être transposée dans d'autres domaines, il prend l'exemple de deux disciplines pratiques, le droit et la médecine. Leurs règles opératoires, dit Leibniz, visent au même idéal de rigueur et de validité universelle que les démonstrations mathématiques, mais elles sont empêchées par la multiplicité de leurs distinctions et de leurs exceptions. Cet état, loin de leur être inhérent, ne fait que marquer l'état primitif de leur avancement :

> Il est vrai qu'il y a des règles [de jurisprudence], qui ont des exceptions, surtout dans les questions où il entre beaucoup de circonstances [...] Mais pour en rendre l'usage sûr, il faut que ces exceptions soient déterminées en nombre et en sens, autant qu'il est possible : et alors il peut arriver que l'exception ait elle-même ses sous-exceptions, c'est-à-dire ses réplications, et que la réplication ait des duplications, etc., mais au bout du compte, il faut que toutes ces exceptions et sous-exceptions, bien déterminées, jointes avec la règle, achèvent l'universalité[74].

Ce que Leibniz exprime de manière parfaitement claire dans ces lignes, c'est l'idée de systématicité qu'il juge analytiquement contenue dans la notion même de règle. Dans un écrit de jeunesse[75], il s'était déjà élevé contre le dicton « il n'y a pas de règle sans exception », démontrant son absurdité par un argument de rétorsion. Ici il conçoit la règle de manière bien plus générale encore, comme contenant ses propres exceptions, mais aussi celles contenues en celles-ci mêmes, c'est-à-dire comme un chemin complet qui permet infailliblement de déterminer la solution recherchée. Ainsi continue-t-il :

> Mais si ces sortes de règles, chargées d'exceptions et sous-exceptions, devaient entrer dans les disputes académiques, il faudrait toujours disputer la plume à la main, en

72. *Projet et Essais pour arriver à quelque certitude pour finir une bonne partie des disputes et pour avancer l'art d'inventer* (Leibniz 1903, p. 175).
73. *Lettre à Jean Gallois*, D.II.1, p. 381, passage plus complet cité plus haut, § 13.
74. *Nouveaux Essais sur l'Entendement Humain*, IV, 7, §11, (Leibniz 1986, p. 331).
75. *Nova Methodus discendae docendaeque Jursiprudentiae* (1667), A.VI.1, 308-310.

> tenant comme un protocole de ce qui se dit de part et d'autre. Et cela serait encore nécessaire d'ailleurs, en disputant constamment en forme par plusieurs syllogismes [et prosyllogismes et proprosyllogismes), mêlés de temps en temps de distinctions, où la meilleure mémoire du monde se doit confondre[76].

Dans ce second paragraphe est clairement exprimée l'idée d'un déroulement déterministe du raisonnement, suivant une règle qu'on peut dire universelle car ses exceptions sont en nombre *fini*, qui nécessite une *mémoire graphique* pour aller à son terme. Même si Leibniz formule l'idée de manière hypothétique, voire incrédule quant à sa praticabilité, il y a là quelque chose qui ressemble beaucoup à ce que nous appellerions l'exécution d'un programme.

Dans cet exemple de la jurisprudence, on atteint enfin, de manière claire, l'idée que la délibération peut être exprimée de manière procédurale, et qu'elle a une affinité profonde avec les démonstrations mathématiques. Écrire une procédure et écrire une démonstration sont essentiellement la même chose – une idée que nous retrouverons lorsque nous étudierons l'équivalence entre preuve et programme. Il semble donc que c'est là que Leibniz se rapproche le plus de notre notion moderne de programmation. Pourtant, d'un autre point de vue, il en reste également infiniment éloigné.

En effet, dans le projet de Caractéristique universelle qu'il promeut, la procédure et la règle ont vocation à se simplifier *jusqu'à devenir entièrement transparentes*. Que la jurisprudence ait tant d'exceptions, et la médecine tant de règles empiriques, ces faits pointent vers leur perfectibilité. Les règles s'abrègent et se simplifient en se généralisant, si bien que, si la langue exprimant directement les notions primitives était trouvée, une procédure *unique* de calcul viendrait remplacer toutes les autres. La langue *universelle* qu'appelle Leibniz de ses vœux ne l'est pas seulement parce qu'elle transcende les frontières et les cultures, mais parce qu'elle peut s'appliquer telle quelle à n'importe quel problème[77]. Une seule procédure, de type grammatical, servirait donc à représenter toutes les autres, de la même manière qu'on peut dériver des règles particulières de règles plus générales. La langue dont rêve Leibniz serait ainsi « très aisée à apprendre[78] ». Ainsi, Leibniz ne s'intéresse pas aux règles et aux procédures en elles-mêmes, dans leur positivité et leur épaisseur propre, telles qu'elles seront étudiées plus tard par Babbage et Turing. Il voit bien qu'elles permettent aux ignorants de résoudre des problèmes,

76. *idem*, p. 331).
77. Voir plus haut, § 14.
78. *Préface à la Science Générale* (1677) (Leibniz 1986, p. 154).

mais cela ne l'intéresse pas. Car avec la Caractéristique dit-il, l'ignorance ne sera plus seulement « prise en charge » : elle sera rendue tout bonnement impossible :

> Les chimères que celui qui les avance n'entend pas ne pourront pas être écrites en ces caractères. Un ignorant ne pourra pas s'en servir, ou s'efforçant de le faire, il deviendra savant par là même[79].

Les règles et procédures doivent donc être bien considérées comme un pis-aller que Leibniz cherche à dépasser grâce à la Caractéristique :

> Si les paroles étaient faites suivant un artifice que je vois possible, mais dont ceux qui ont fait des langues universelles ne se sont pas avisés on pourrait arriver à cet effet par les paroles mêmes, ce qui serait d'une utilité incroyable pour la vie humaine. Mais en attendant il y a un autre chemin moins beau, mais qui est déjà ouvert, au lieu que l'autre devrait être fait tout de nouveau. C'est en se servant de caractères à l'exemple des mathématiciens, qui sont propres de fixer notre Esprit, et en y ajoutant une preuve des nombres[80].

Si la possibilité d'une connaissance procédurale est donc clairement conçue par Leibniz, elle n'en est pas le modèle ; elle est un détour pour pallier la confusion de notre pensée lorsqu'elle ne peut parcourir immédiatement une longue chaîne d'éléments. La pensée claire doit cependant lui être préférée dès que possible :

> [C'est une faute de] vouloir réduire les vérités aux maximes […] à contretemps et sans besoin, car l'esprit humain envisage beaucoup tout d'un coup et c'est le gêner que de le vouloir obliger à s'arrêter à chaque pas qu'il fait et à exprimer tout ce qu'il pense. C'est justement comme si en faisant son compte avec un marchand ou avec un hôte, on le voulait obliger de tout compter avec les doigts pour en être plus sûr[81].

Le caractère séquentiel des procédures n'a plus lieu d'être dans la connaissance intuitive, qui envisage tout d'un coup. On retrouve ici un lointain écho du rapport décrit par Keckermann entre le système extérieur et l'*habitus* intérieur, le premier étant simple précepte pour le développement du second, rapport qui lui-même évoque celui entre la méthode artificielle et la méthode naturelle chez La Ramée. Sur ce point, bien qu'on puisse à bien d'autres égards faire de lui un des « logiciens dont les idées rendit [l'informatique] possible[82] », Leibniz reste solidement ancré dans la rationalité baroque[83] des Temps modernes, en ce qu'il croit à la possibilité

79. *Lettre à Jean Gallois* D.II.1, p. 381.
80. *Projet et Essais pour arriver à quelque certitude...* (Leibniz 1903, p. 175).
81. *Nouveaux Essais sur l'Entendement Humain*, IV, 7, §11, (Leibniz 1986, p. 332).
82. (Davis 2012, p. xiv).
83. (Knecht 1981).

d'une « clé » donnant accès au savoir universel.

§ 116. Le découplage des systèmes et des problèmes

La position de Leibniz permet de rendre patente la difficulté qu'ont les modernes à concevoir proprement la relation entre un système (qu'il soit scientifique ou technique) et la résolution de problème qu'il rend possible.

D'un côté il y a la routine sans système, particulière et irrégulière, où le jugement est asservi à l'habitude et à la coutume. D'un autre il y a le système accompli tel que le décrit Leibniz, comme celui de la jurisprudence, qui guide si bien le jugement que celui-ci n'a plus qu'à fournir les faits et acquiescer à l'enchaînement des syllogismes jusqu'à la résolution finale du problème. Entre ces deux pôles où la réflexion proprement dite semble annihilée, l'idée moderne de méthode oscille : elle doit certes éveiller et diriger l'esprit, cependant sans que ce guidage devienne pure contrainte. Mais comment décrire alors la contribution propre de l'esprit, si celle-ci doit échapper aux règles aussi bien qu'à l'arbitraire ?

Les *Règles pour la conduite de l'esprit* donnent une réponse assez directe dans le cas des mathématiques : la réflexion se meut dans le cadre donné par la géométrie, qui fournit aussi bien les termes du problème à résoudre que ceux avec lesquels la solution pourra être composée. La géométrie, comme système de connaissance, ne fait rien de plus, et c'est d'ailleurs pour cela que les *Règles* doivent servir d'heuristique pour guider l'esprit dans sa recherche.

Lorsqu'elle visent l'exhaustivité, les tables et les dichotomies des réductions en art cherchent à résoudre *directement* les problèmes rencontrés : ainsi, par exemple, les « tables de quadrature de bataille qui permettent de connaître instantanément, selon les effectifs dont on dispose, la meilleure répartition des soldats dans l'espace[84]. » Lorsqu'au contraire elles se contentent d'énumérer des considérations pertinentes sur les problèmes, elles échouent à les poser de telle manière qu'ils puissent être résolus systématiquement. Ce que manquent donc de faire la plupart des réductions en art, c'est de créer un espace permettant de *poser* des problèmes en des termes suffisants pour qu'ils puissent décrire adéquatement la diversité et la complexité des situations à affronter, et cependant en nombre assez restreint afin que leur manipulation puisse en être apprise.

Autrement dit, le défaut de ces méthodes est de ne savoir *découpler* proprement les systèmes de règles qu'elles construisent des problèmes que ces systèmes

84. (Dubourg Glatigny et Vérin 2008, p. 61).

doivent permettre de résoudre. *Découplage* signifie que le système, en tant que tel, n'a jamais pour vocation centrale de donner la solution d'aucun problème. Cela n'implique pas pour autant indifférence et abstraction : il doit donner les moyens appropriés de les décrire *tous* et de les résoudre *démonstrativement*, un cahier des charges formidable s'il en est.

La géométrie est à l'époque moderne le modèle d'un tel système. Opérant seulement avec la règle et le compas, elle se donne à elle-même ses problèmes – des théorèmes à prouver ou des figures à construire[85] et les moyens de les résoudre. D'ailleurs, la découverte que ceux-ci étaient insuffisants à construire toute figure, comme par exemple la trisection d'un angle, fut décisive pour sa révolution cartésienne[86].

L'idée de fonder des réductions en art sur des principes géométriques apparaît assez tôt aux Temps modernes. Elle apparaît par exemple, de manière surprenante, dans les manuels d'escrime du XVI^e^ siècle[87]. Il s'agissait sans doute, pour les maîtres bretteurs, comme pour les maîtres de tant d'autres arts dits traditionnellement *mécaniques*, de faire reconnaître l'escrime comme art libéral. On peut sans doute évoquer le néo-platonisme ambiant des sphères cultivées de l'époque, qui promouvait naturellement la géométrie comme modèle de la connaissance. La voie des mathématiques mixtes semblait en tous les cas plus plausible dans cette démarche concernant une discipline corporelle que celle des arts du *trivium* (logique, grammaire et rhétorique), même s'ils furent également sollicités. Mais au-delà de ce contexte socio-culturel, il y avait dans ces recherches une intuition importante : la géométrie donnait l'idée de fonder un art sur un langage opératoire uniforme de description et de résolution des problèmes, qui permettait de surcroît leur validation démonstrative. Ainsi, de nombreux manuels d'escrime du XVI^e^ siècle sont animés par l'obsession de réduire les gestes à l'enchaînement de positions élémentaires, les gardes – un « alphabet » dit même l'un des auteurs : l'art de l'escrimeur serait ainsi réduit à connaître les lois de ces enchaînements, relativement à celles de l'adversaire[88].

85. En géométrie classique, les *problèmes* désignent spécifiquement les demandes de construction de figures, par opposition aux *théorèmes* qui demandent d'en prouver des propriétés. Cette distinction nous intéressera plus loin dans notre réflexion, mais dans ce chapitre nous employons *problème* dans le sens général qu'il a aujourd'hui, et qui recouvre toute demande.
86. Voir (Bos 2001), et plus loin § 124.
87. (Brioist 2008).
88. (ibid., p. 240-241).

Ce sont cependant les arts de la construction qui peuvent naturellement le mieux s'inspirer de la géométrie, et c'est là que le découplage entre conception et exécution peut être le mieux visible, avec l'apparition au XVII^e siècle de la figure centrale de l'ingénieur moderne et de son « chef d'œuvre », le devis. L'étude de cette évolution est l'objet de la section suivante.

7.3 Ingénieurs, division du travail et machines

Nous commençons par mettre en exergue le rôle majeur que joue la géométrie dans la transformation de l'art des ingénieurs au XVII^e siècle, tel que l'a analysé Hélène Vérin. Il ne s'agit pas seulement d'améliorer la conception et le dessin des plans des ouvrages, mais de subordonner l'ensemble du projet à une démarche rationnelle de part en part, depuis la qualification des besoins jusqu'à l'exécution des travaux (§ 117). Dans le *devis* de l'ingénieur, pièce maîtresse du projet, on voit le découplage entre conception et exécution devenir une véritable *division du travail* intellectuel à de multiples niveaux (§ 118). Cette perspective nous permet de mettre en évidence le rôle très spécifique que joue l'idée de *machine* dans ce mouvement (§ 119).

§ 117. Les ingénieurs et les devis

Selon Hélène Vérin, la réduction en art de la fortification, au XVI^e siècle, consiste à

> proposer, non seulement les connaissances préalablement utiles, mais aussi les moyens de remonter des cas concrets qui sont imposés aux praticiens lorsqu'ils doivent « inventer » la fortification d'une place donnée, vers des types de solutions connues et maîtrisées[89].

Il s'agit, en particulier, de bien poser le problème en étudiant les forces et les faiblesses naturelles du site en terme de défense relativement aux moyens offensifs dont l'ennemi pourrait disposer. Par ailleurs, les formes typiques d'une fortification sont décrites et discutées, ainsi que les éléments « standards » dont elle est constituée (courtines, terre-pleins, etc.)[90]. En termes de moyens de résolution, on conseille la simulation (imaginer une attaque de la forteresse), on fournit des tables montrant le résultat d'*essais*, c'est-à-dire d'expérimentations d'effets spécifiques

89. (Vérin 1993, p. 137).
90. (ibid., p. 138-141).

(par exemple des tirs de canons sur une muraille, suivant différents paramètres), on donne la solution optimale à certains problèmes spécifiques (comme la détermination de l'angle optimal d'une muraille), et enfin on illustre des solutions globales possibles par des dessins et plans, qui deviennent de vrais supports de la réflexion[91].

Il s'agit donc de *guider* la résolution de problème, en aidant l'investigateur à bien le décrire, dans toutes ses dimensions pertinentes, et à organiser l'espace des solutions en options prédéfinies et partiellement optimisées qu'il n'aurait plus qu'à assembler, selon les procédures heuristiques recommandées. Si la géométrie est bien présente dans ses traités, elle a un rôle heuristique et instrumental.

Le modèle élaboré par Galilée dans sa *Brève introduction à l'architecture militaire*, caractérisé par « une extrême abstraction [et] la prééminence de la géométrie[92] » va marquer une inflexion importante. Au XVII^e^ siècle, l'ordre d'exposition des sujets s'inverse et les traités débutent de plus en plus souvent par des exposés de géométrie. Il ne s'agit plus seulement de rappeler des notions pratiques concernant l'art de mesurer et de déterminer les proportions, mais c'est l'art de l'ingénieur lui-même qui est conçu comme celui d'une construction d'abord géométrique, partant de principes assurés, aux effets démontrables, qu'il faut ensuite patiemment accommoder aux accidents du sites, grâce à la mesure précise et aux instruments[93].

Cette évolution va en préparer une autre, décisive pour notre propos, celle du métier d'ingénieur qui occupe à partir de la fin du XVII^e^ siècle un « lieu inédit », selon Hélène Vérin, une place « où l'on ne s'était jamais avisé de mettre personne », selon le mot de l'*Encyclopédie Méthodique* qu'elle cite[94]. Le mot d'*engignour* sont attestés dès le Moyen Âge. Il désigne le constructeur d'*engins* de sièges, et il est donc d'abord un charpentier, mais devient fondeur lorsqu'apparaît l'artillerie et architecte quand les fortifications deviennent cruciales dans la pratique de la guerre. *Engin*, pour sa part, connote avant tout la ruse et le détour ; la notion s'oppose de manière insistante à la force. Pour les arabes, la science de l'ingénieur est bien la mécanique antique, qu'ils interprètent comme les *ruses de la géométrie*. L'association aux mathématiques est donc ancienne[95].

91. (Vérin 1993, p. 152-166).
92. (ibid., p. 141).
93. (ibid., p. 167-178).
94. (ibid., p. 181).
95. (ibid., ch. 1 et 2). Voir également (Pautet 2018). Sur l'interprétation de la mécanique, voir

Au Grand Siècle, l'ingénieur se situe à l'intersection d'un « champ de forces » diverses : l'ingénieur peut être militaire, commis de l'État, entrepreneur, savant, technicien – et parfois tout cela à la fois. Il lui revient proprement de *dénouer* ces confrontations d'intérêts utilitaires variés, économiques et régaliens en imposant une *conception* qui puisse satisfaire toutes les parties prenantes. L'apparition de l'ingénieur moderne est en particulier indissociable, en France, de l'émergence de l'État moderne qui cherche à accroître sa puissance économique et militaire par des politiques raisonnées de constructions militaires, navales, civiles[96]. L'ingénieur, en tant qu'il est commis de l'État, occupe alors un rôle unique et nouveau : élaborer d'une part un projet qui puisse convaincre différents ministères de l'optimalité de la conception relativement à des critères variés et souvent contraires – potentiel de défense, coût, praticalité, etc. – d'autre part mettre en concurrence des entreprises de manière transparente et équitable, veiller à choisir la meilleure offre, contrôler la conduite des travaux. Tout cela converge vers ce que Bélidor (1698 - 1761), le grand ingénieur militaire, qualifie de « chef d'œuvre de l'ingénieur », le *devis*, qui décrit le projet, démontre ses propriétés, détaille les étapes possibles de sa construction, estime son coût[97].

Le devis doit répondre à toutes ces finalités et convaincre toutes les parties prenantes de la possibilité et de l'optimalité du projet proposé. Il est, tout autant qu'un plan, une démonstration, voire une argumentation qui doit également persuader, ou même séduire. Dans le devis s'affirme, comme le dit Vérin, « un souverain triomphe de l'intelligible sur le sensible ».

La géométrie est bien entendu centrale dans le projet. À la lecture du devis,

> la structure géométrique de l'artifice, même si elle a été longuement réfléchie après de nombreuses mesures et mûres délibérations, par essais retouchés, repris sur le papier, jusqu'à être judicieusement adaptée au local, n'en apparaît pas moins […] comme l'irruption d'un espace essentiellement autre, […] entièrement conçu qui, se surimposant au souvenir du lieu et aux jugements qui s'y accordaient, met, en quelque sorte, ces jugements en suspens[98].

L'inspiration géométrique ne s'arrête pas là. En amont de la conception, les mathématiques servent également à décrire les contraintes fonctionnelles du problème à résoudre – comme par exemple la proportionnalité entre la valeur défensive

également plus haut, § 111.

96. (Vérin 1993, p. 184-188).
97. (ibid., p. 227).
98. (ibid., p. 233).

d'une place forte et le coût de sa construction[99]. En aval, elles déterminent le détail des constructions, les matières à utiliser, en fonction de leurs propriétés, ainsi que l'ordre de la construction :

> On est donc passé des dimensions aux matières, de l'ordre général au détail de l'ordre particulier […] ; puis de l'ordre de toutes les parties à celui de leur construction. Ce dernier applique une méthode générale. De part en part, l'abstraction – géométrique, méthodique – règle sous le terme de « détail » la diversité des matières et des conduites. […] L'établissement du niveau général, les profils des revêtements, leurs épaisseurs, la profondeur des fondations ont été décidés après que l'ingénieur eut reconnu le terrain par des sondes, eut apprécié la qualité des terres[100]…

Comme le remarque Vérin, une telle démarche, qui exclut les entrepreneurs contractants de toute participation à l'élaboration du devis, et les oblige à y acquiescer en bloc, ne brille pas toujours par son efficacité. Au-delà des particularités historiques des procédures d'adjudication de ces marchés publics aux XVII^e^ et XVIII^e^ siècles, un point essentiel est cependant conquis ici : l'établissement d'un langage commun, sinon standardisé, des opérations de construction, qui permet à l'ingénieur de disposer *a priori* du savoir des opérations et des figures réalisables, ainsi que de leurs propriétés (coûts, temps, conditions) :

> La généralisation du contrat d'entreprise pour la production d'artifices exigeait une autre organisation sociale des savoirs utiles, qui correspondait à un autre découpage de leur acquisition et de leur pratique, […et qui permettait] de disposer, d'une part, d'ouvriers informés, capables de lire et d'exécuter un devis, et d'autre part, de concepteurs et directeurs de projets, chargés d'en dresser les devis. […] La particularité du contrat, c'est qu'il ne s'agit pas simplement d'anticiper, mais de se mettre d'accord sur une anticipation. Or on ne se met d'accord qu'en usant de mots, et, dans le cas du contrat, sur un texte[101].

Il s'agit là d'un vecteur important, selon Hélène Vérin, de l'apparition de la technique moderne. Bien plus que les réductions en art, le contrat oblige « la tâche à devenir opération[102] », et ces opérations à se normaliser et à se doter d'un vocabulaire technique uniforme. Cette normalisation fait « coup double ». D'une part elle rend l'ingénieur capable « de faire exécuter les travaux les plus difficiles », si bien que « rien ne lui échappera » et qu'il « ira même jusqu'à prévoir les accidents

99. (Vérin 1993, p. 368-378).
100. (ibid., p. 234).
101. (ibid., p. 229).
102. (Dubourg Glatigny et Vérin 2008, p. 58).

qui pourraient survenir »[103]. D'autre part, elle permet la divulgation et la diffusion du savoir secret des ateliers – préparant ainsi la voie aux encyclopédistes.

Toutes ces observations nous intéressent car il n'est pas difficile d'y reconnaître les éléments essentiels du modèle contractuel de la programmation[104]. Le devis est une spécification du travail à accomplir, dont la relation avec les besoins du maître d'ouvrage doivent être démontrés, et dont la réalisation par l'entrepreneur doit être possible. C'est là que se réalise le *découplage* entre, d'une part, le système socio-technique normalisé qui rend possible aussi bien certains problèmes que certaines solutions, et d'autre part le lieu et le moment de sa mobilisation pour l'élaboration particulière d'un problème et sa solution. Une résolution de problème n'est pas systématique car elle serait *donnée* par le système, telle une recette qui y serait consignée, mais parce qu'elle est construite *dans les termes* du système, qui donne par là les moyens de s'assurer de sa sureté.

La géométrie a permis, historiquement, de montrer la possibilité d'une telle organisation de la technique. Bien entendu, celle-ci n'est nullement limitée aux disciplines qui pourraient avoir l'usage de la géométrie euclidienne. Nous reviendrons cependant sur son importance paradigmatique à la section suivante – cette fois concernant le savoir théorique et, plus loin encore, au chapitre 9 où nous tenterons de cerner la raison de cette importance – sa constructibilité.

§ 118. Technique et division du travail

L'organisation moderne du projet qui se met en place autour du devis de l'ingénieur est celle d'une triple *division du travail*. Il y a d'abord celle de l'organisation même des travaux, que permet la normalisation des opérations évoquée ci-dessus. S'il n'y a peut-être pas encore l'approche systématique de l'industrie dans la répartition des tâches[105], celle-ci est à présent possible en droit. Mais il y a également *une seconde* division du travail, d'un niveau supérieur, qui a lieu entre l'ingénieur et le contractant, entre la phase de conception et la phase d'exécution du projet. Ici se distinguent clairement les deux formes de règles que leur prodigue le système technique. D'une part, il y a les règles normalisées de l'exécution elle-même, qui sont leur langage commun ; elles indiquent quelles opérations sont réalisables, à quel coût, etc. Elles permettent notamment à l'ingénieur de se concentrer sur la

103. Nous paraphrasons ici VÉRIN (1993, p. 229) citant Bélidor.
104. Voir plus haut section 3.3.
105. Voir plus haut section 5.3.

conception pure de son ouvrage. Mais il y a également, d'autre part, les règles de l'art de bien concevoir, c'est-à-dire les préceptes de la méthode que doit suivre l'ingénieur pour réussir son projet. L'ambiguïté que nous avions perçue dans les réductions en art sur la fonction des règles – guide ou contrainte du jugement – est ici clairement résolue par la division du travail : dans la phase de conception se concentre toute la dimension libérale de l'art, domaine propre de l'esprit qui ne peut qu'être guidé, et non contraint par les règles. Dans la phase d'exécution se concentre sa dimension mécanique et servile, qui exige d'être normalisée et dirigée exactement. Il est à noter – nous l'avons suffisamment vu – qu'une telle distinction est seulement relative, et qu'elle peut se présenter récursivement : on peut être contractant d'un côté et maître d'ouvrage de l'autre, car ce qui apparaît être une opération simple d'un côté peut se révéler être un vrai problème à résoudre de l'autre. Enfin, si on distingue à présent maître d'ouvrage, ingénieur et contractant, on retrouve dans cette triade le triple *ordonnancement* par lequel Thomas d'Aquin décrivait la technique, celle de la finalité, de la structure (que conçoit l'ingénieur, et dont l'ensemble du devis découle) et de la construction[106].

Il apparaît enfin une *troisième* division du travail, d'un niveau encore supérieur, entre savant et ingénieur. Cette fois c'est l'investigation elle-même qui se dédouble en résolution de problème d'une part et en investigation systématique de l'autre, l'une qui utilise le système, l'autre qui l'élabore – cette distinction que Ryle avait jugé centrale pour la notion de *théorie*. Nous argumenterons plus tard que c'est là seulement que peut être fondée une distinction stable entre l'investigation théorique et l'investigation pratique[107].

Ces divisions du travail à de multiples niveaux, d'abord entre contractants de spécialités différentes, puis entre maître d'ouvrage, ingénieur et contractant, et enfin entre savant et ingénieur, s'articulent et se croisent de manière complexe au sein d'un projet. Un ingénieur dirige de nombreux contractants, peut faire appel à plusieurs expertises théoriques, et doit souvent répondre à plusieurs clients aux objectifs différents. Mais il peut aussi y avoir plusieurs ingénieurs sur un projet, certains organisés hiérarchiquement, d'autres spécialisés sur une dimension du projet, d'autres encore intervenant à des moments précis, etc. Tous, cependant, agissent selon un certain *ordre* de leurs préoccupations particulières, « celui du traité (enseignement de la discipline) ; celui du projet (conception de l'ouvrage) ;

106. Voir plus haut, § 117.
107. En particulier section 8.2.

celui du devis (contrat avec des entrepreneurs) ; celui de la réalisation (conduite des travaux) » dit Hélène Vérin[108], à quoi on pourrait ajouter au moins celui de l'usage (« mode d'emploi ») et celui de la dépense (financement). La technique moderne est l'emboîtement et l'organisation d'*ordonnancements* divers, en nombre indéfini, à partir des trois schémas élémentaires que nous avons décrits.

C'est là qu'apparaît aussi la possibilité d'un système technique – qui n'est pas seulement un système de règles, mais également un système de moyens humains, matériels et financiers qui *rendent possible* certains problèmes et certaines solutions. C'est au sein d'un tel système, comme nous l'annoncions en conclusion du chapitre précédent, que peut être résolu le paradoxe (en tous les cas pour Aristote) d'un savoir technicien qui serait systématique bien qu'il reste entièrement contingent, du fait de sa dépendance aux intentions humaines. C'est parce que ces intentions elles-mêmes peuvent se constituer en *systèmes de fins et de moyens*, comme le voit très bien Lambert dans les années 1760[109], qu'un savoir systématique et démonstratif de la technique devient enfin possible, et qu'on peut commencer à parler de *sciences de l'ingénieur* et de *technologie*.

§ 119. La spécificité des machines

La différence qu'il y a bien sûr entre cette conception et le modèle contractuel que nous avons donné de la programmation, est que la réalisation de la spécification, dans le second cas, est confié à une *machine*. Quelle est l'importance de cette différence ? Qu'est-ce qui distingue, parmi toutes les constructions, celles qui sont confiées à des machines ?

Une remarque de Dubourg Glatigny et Vérin peut ici nous éclairer. Elle introduit leur réflexion déjà citée concernant l'ordonnancement des jugements auquel procède la réduction en art[110]. On peut affirmer, selon eux, que la machine est l'« idéal » de la réduction en art :

> [Selon] la remarque que fait Galilée au début de ses *Mécaniques*, contrairement à ce que croit le vulgaire, la machine n'économise pas des forces mais du jugement. […] Les dispositifs mécaniques évitent tout ce qu'impliquent de tâtonnements, de réflexions, de jugements et de décisions les gestes à faire et leur succession. La machine est […] le modèle idéal d'une action qui réduit au minimum l'obligation de

108. (Vérin 1993, p. 245).
109. (Lambert 1764, p. 134-137). Sur Lambert, voir plus loin, § 183.
110. Voir plus haut, p. 454.

faire des choix[111].

On peut être surpris de l'assertion que la réduction en art vise la *mécanisation*, alors qu'elle procède le plus souvent, comme nous l'avons vu, d'une entreprise d'éducation et de développement de l'esprit. Et, quelques lignes plus loin dans le texte déjà cité, les auteurs affirment d'ailleurs que « l'objectif [de la réduction en art] n'est pas de supprimer l'exercice du jugement mais de le guider, de l'abréger, de le faciliter. »

On se retrouve donc là au cœur de l'ambiguïté analysée dans la section précédente, entre la visée qu'a la méthode de guider le jugement et sa tentation de le contraindre. L'ambiguïté se résout lorsqu'on se réfère au texte de Galilée évoqué par les auteurs :

> La troisième utilité des Machines est très grande, parce que l'on évite les grands frais et le coût en usant d'une force inanimée, ou sans raison, qui fait les mêmes choses que la force des hommes animée, et conduite par le jugement, comme il arrive lorsque l'on fait moudre les moulins avec l'eau des étangs, ou des fleuves, ou avec un cheval qui supplée la force de 5 ou 6 hommes. Et parce que le cheval a une grande force, et qu'il manque de discours, l'on supplée le raisonnement nécessaire par le moyen des roues et des autres Machines qui sont ébranlées par la force du cheval, et qui remplissent et transportent le vaisseau d'un lieu à l'autre, et qui le vident suivant le dessein de l'ingénieur[112].

Le jugement que la machine économise est celui des hommes de main, préposés à des tâches serviles, et dont la division du travail entre conception et exécution, décrite plus haut, vise à se passer entièrement. Elle augmente au contraire le jugement requis du directeur de travaux, qui doit construire la machine ou la configurer pour le travail à effectuer. La machine permet de transférer le jugement résiduel qu'on trouve encore dans l'exécution vers la conception – à rendre plus nette encore cette division du travail – ce qui permet, en outre, de concevoir le processus de travail ainsi automatisé dans sa nécessité logique propre, comme le remarquera Marx[113].

La machine apparaît ici étroitement liée à la résolution de problème, voire : sa forme pure et rigoureuse. Si on définit en effet la géométrie comme la mathématique qui construit ses objets et en démontre les propriétés[114], alors la mécanique, prise comme géométrie des machines[115], permet d'établir des liens nécessaires entre

111. (Dubourg Glatigny et Vérin 2008, p. 62).
112. (Galilei [1634] 1966, p. 25).
113. Voir plus haut, § 86.
114. Voir plus loin, chapitre 9.
115. Voir la section suivante.

les trois ordres qu'avait distingués Thomas, celui de la fin, de la structure et de la construction. Les propriétés de la machine sont en effet ses mouvements, qui réalisent sa fonction, comme la rotation de la meule fait la mouture du blé. La résolution de problème est ici rigoureuse car il devient possible de *démontrer* que la fonction demandée est accomplie par la structure présentée, et de *montrer* comment construire celle-ci. Il peut être démontré que la rotation de la meule suivra celle des engrenages entraînés par les ailes, si le moulin est construit de telle façon et avec telle méthode. Aussi n'est-ce pas d'une manière seulement métaphorique qu'il faut entendre la célèbre exclamation de Diderot face au métier à bas :

> Le métier à faire des bas est une des machines les plus compliquées et les plus conséquentes que nous ayons : on peut la regarder comme un seul et unique raisonnement, dont la fabrication de l'ouvrage est la conclusion ; aussi règne-t-il entre ses parties une si grande dépendance, qu'en retrancher une seule, ou altérer la forme de celles qu'on juge les moins importantes, c'est nuire à tout le mécanisme[116].

Diderot ne fait ici que reprendre l'association faite déjà par Galilée dans le texte cité plus haut, entre mouvement mécanique et raisonnement, une association au fond banale, puisqu'elle est véhiculée depuis l'Antiquité par l'image fascinante de l'automate, dont les rouages miment l'action raisonnable. Au fond de cette image, il n'y a pas une métaphore, mais de la géométrie.

Ici nous sommes en mesure d'énoncer pour la première fois une hypothèse quant à la nature de cette approche *systématique* de la technique que nous avons vue à l'œuvre dans l'industrie et dans l'informatique. Elle consisterait à prendre pour objet, non pas les artefacts fabriqués par les personnes humaines, mais directement la relation nécessaire qui doit exister entre deux moments de l'action humaine planifiée, l'usage d'une part (ou plus généralement l'exécution d'un plan), et la construction de l'autre (ou plus généralement la planification) – ces deux moments n'étant que parfois médiatisés par une structure instrumentale concrète, l'artefact. Cette manière de voir permettrait d'abord de rendre compte de toutes les techniques qui, à proprement parler, se passent d'artefacts pour accomplir leurs effets – comme l'action du médecin vise directement à rétablir la santé chez le patient, et la conduite du navire par le pilote vise directement à amener les passagers à bon port. Cependant là n'est pas l'essentiel ; ce qui serait nouveau ici c'est l'idée, saugrenue pour Aristote, qu'il pourrait exister des relations « géométriques » entre ces deux séries de phénomènes, qui permettraient de passer de l'une à l'autre par

116. *Encyclopédie*, article « Bas », vol. 2, p. 98a-113b.

des raisonnements rigoureux et exhaustifs.

On peut ici aller encore plus loin et anticiper quelques remarques que nous développerons au chapitre 11. Produisant un mouvement et produites par un mouvement, une machine peut en construire d'autres – il s'agit d'une notion qui admet la récursivité. Cela n'est certes pas suffisant pour en faire une propriété intéressante : si la machine était déterminée à accomplir un seul mouvement, elle ne pourrait engendrer qu'une seule autre figure de machine, et on pourrait tout aussi bien produire celle-ci directement. Cependant, lorsqu'on construit une machine, on y introduit en général des *degrés de liberté*, par exemple entre diverses positions ou vitesses possibles. Grâce à ses degrés de liberté, la machine est capable de plusieurs mouvements et donc également de plusieurs démonstrations, elle devient configurable selon des règles qui lui sont propres. Elle peut devenir un *atelier* où une diversité d'autres machines seront assemblées ; elle est un système de règles qui peut en générer d'autres.

7.4 La machine, catégorie d'intelligibilité

À la fin de la section précédente, la machine nous est apparue comme une catégorie centrale d'intelligibilité des systèmes techniques dont la fin des Temps modernes tracent les premières esquisses.

Or il s'agit là d'un concept qui, comme ceux d'*ingénieur*, de *méthode*, de *système*, évolue rapidement aux Temps modernes. Dans cette section et la suivante, nous nous intéressons au rôle proprement *épistémique* qu'il a pu également avoir à cette époque dans l'émergence de la mécanique comme science centrale de la nature. Il apparaît ainsi comme un des lieux essentiels où se laisse comprendre l'unité des savoirs théoriques et pratiques.

Nous rappelons d'abord quelques éléments historiques concernant l'invention des machines et leur réception à la fin du Moyen Âge, puis leur rôle dans l'émergence de la « philosophie mécanique » désignant, à partir de Boyle, le programme de la nouvelle science de la nature (§ 120).

Nous examinons dans la suite de cette section la position classique de Koyré et Dijksterhuis sur la question posée. Selon eux, le rôle de l'idée de *machine* dans l'avènement de la mécanique comme science centrale de la nature doit être minimisé. Cet emploi est d'abord métaphorique et, au mieux, il a permis de concevoir la possible mathématisation des phénomènes de la nature. En réponse à cette po-

sition, nous montrons d'abord que la métaphore de la machine est ambivalente, puisqu'elle est utilisée dès le XVI^e siècle par Marsile Ficin pour faire l'apologie des causes finales contre le mécanisme atomiste de Lucrèce (§ 121). Nous étudions ensuite le lien entre machine et mathématisation qui, selon Koyré, réside dans l'idée de la possibilité d'une *mesure précise* des phénomènes naturels. Nous argumentons au contraire, dans la continuité de nos réflexions précédentes, que la machine est d'abord une catégorie d'intelligibilité, qui permet la transposition de la manière géométrique de résoudre les problèmes aux phénomènes dynamiques de la nature (§ 122).

§ 120. Machines et mécanique

La mécanique eut dès l'Antiquité un statut double entre théorie et pratique, bien avant qu'Al-Farabi remarque qu'il en était ainsi de toute discipline du savoir[117]. Aristote cite la mécanique (*τά μηχανικά*), aux côtés de l'astronomie, de l'harmonique et de l'optique, parmi les sciences mathématiques mixtes[118]. Si on suit le traité des *Mécaniques* du pseudo-Aristote qui, quel que soit son auteur, est le plus ancien dont nous disposions sur cette discipline, la mécanique porte principalement sur les moyens permettant de soulever et de transporter des poids avec efficacité (la poulie, le levier etc.), l'usage efficace de la force (*pourquoi une pierre lancée avec une fronde va-t-elle plus loin qu'à main-nue ? pourquoi les rameurs au centre du navire ont-ils plus d'efficacité ?*), mais aussi l'explication de certains phénomènes naturels par les causes efficientes (*pourquoi les galets au bord de la mer sont-ils ronds ?*). En tant que sa racine *μηχανάομαι* désigne la compétence pour élaborer des stratagèmes (nos « machinations ») et résoudre des problèmes de manière efficace, la mécanique est donc l'art qui « fait déjouer » la nature au profit de l'homme[119], et qui enseigne à fabriquer des machines et des instruments en général. Qu'il puisse y avoir une mathématique de ces objets techniques n'est pas un problème d'un point de vue aristotélicien, puisqu'Aristote lui-même avait rapproché la délibération de l'investigation géométrique[120]. Elle doit par contre être entièrement distinguée de la physique, qui étudie les causes naturelles, qui sont

117. Voir plus haut, § 111.
118. *Mét.* M.3, *APo.* I.13.
119. Au sujet du mouvement « contre-nature », voir (Roux et Festa 2001) qui examinent ses diverses interprétations à l'époque moderne, jusqu'à Galilée.
120. Voir plus haut, § 96.

finales et nécessaires. La physique, dit le célèbre prologue des *Mécaniques*, fournit seulement son *objet* à la mécanique, mais ce sont les mathématiques qui étudient ses dispositifs et ses méthodes[121], autrement dit les manières de détourner le mouvement issu de causes naturelles en vue d'une cause finale intentionnelle. En termes contemporains, nous dirions que la mécanique étudie les possibilités d'intervention de la personne humaine dans le cours de la nature.

Au Moyen Âge, comme déjà évoqué, le terme *mécanique* ne désigne plus spécifiquement cette science des poids et des machines[122]. Les « arts mécaniques », selon l'usage consacré par Hugues de Saint-Victor, désignent en général les arts utiles à la vie, ce qui inclut également l'agriculture, l'habillement, etc. Ces « arts serviles » sont alors opposés aux « arts libéraux » du *trivium* et du *quadrivium*. La mécanique ne regagne son statut de science qu'à la redécouverte, au milieu du XVI^e^ siècle, des traités d'Archimède et d'Héron d'Alexandrie, mathématiciens et ingénieurs à la fois, et du traité du pseudo-Aristote. Les ingénieurs des Temps modernes, comme Léonard, Brunelleschi, Galilée, renouent avec cette tradition et « cherchent dans les ouvrages d'Euclide, d'Archimède, de Vitruve ou d'Héron d'Alexandrie une réponse à leurs questions[123] ». Une réflexion profonde s'engage concernant le statut du mouvement étudié par la mécanique[124] : est-il « contre » nature, comme la doctrine aristotélicienne le soutenait ? ou est-il naturel lui-même, puisqu'il se laisse expliquer par des principes mathématiques ? La formation d'un consensus en faveur de la seconde hypothèse, en particulier grâce à Galilée, permit à la mécanique de devenir légitime pour expliquer les mouvements naturels. Le rejet des causes finales et des formes substantielles acheva de la mettre au centre des investigations scientifiques concernant les phénomènes naturels[125].

Quel rôle joua le concept de *machine* dans cette mutation ? D'abord, il faut savoir de quoi l'on parle. Une étude historiographique de Popplow (2007) sur le terme *machina* montre qu'il est utilisé quasi-exclusivement dans l'Antiquité latine

121. *Mécaniques*, 847a26 : *τὸ μὲν γὰρ ὡς διὰ τῶν μαθηματικῶν δῆλον, τὸ δὲ περὶ ὃ διὰ τῶν φυσικῶν*. Winter traduit : « Le *comment* (*the how*) est clair par les mathématiques, le *quoi* (*the what*) par la physique ».
122. (Laird et Roux 2008, p. 5). Voir aussi plus haut § 111.
123. (Rossi [1962] 1996, p. 19). Voir aussi pp. 34-42.
124. Voir à ce sujet (Roux et Festa 2001).
125. Au sujet de la place de la mécanique au sein de la « révolution scientifique » des Temps modernes, nous nous sommes entre autres appuyés sur les ouvrages suivants : (Dear 2019 ; E. J. Dijksterhuis 1986 ; Garber et Roux 2013 ; Koyré 1961 ; Palmerino et al. 2004 ; Rossi [1962] 1996 ; Westfall et Torchin 1977).

pour désigner, soit les machines simples (le coin, le levier, le plan incliné, etc.) étudiées par la mécanique comme science, soit les échafaudages et les machines de guerre, c'est-à-dire des structures composées de manière complexe qui connotent la stabilité. Ce n'est qu'à partir du début du XVI^e siècle que son sens s'enrichit pour désigner également les machines à transmission de mouvement (ou *cinématiques* ainsi que les appelle Sophie Roux[126]), comme les moulins et les automates, principalement les horloges[127]. Tous ces objets sont en effet inventés ou largement perfectionnés et diffusés au cours du Moyen Âge, et c'est d'abord par similarité à l'admiration que Vitruve trouve aux machines anciennes que les ingénieurs des Temps modernes les dénomment du même terme. Ces nouveaux objets vont en effet fasciner l'opinion savante, noble et populaire, comme on le voit en particulier dans la diffusion des *théâtres de machines*, ces ouvrages abondamment illustrés et rédigés par des ingénieurs voulant faire connaître leur savoir-faire[128].

Au XVII^e siècle, le terme est si populaire qu'il devient une expression fourre-tout, comme le remarque un contemporain :

> Nous voyons que cela nous a fait grand plaisir d'avoir de ces mots à la mode, parce qu'il s'en trouve même qui tout seuls signifient tout ce qu'on veut. Il y a quantité de Gens qui lors qu'ils ne peuvent exprimer quelque chose par un mot propre, usent du mot de Machine, ils disent, il faut faire des machines pour cela, & que ce sont là des Machines[129].

§ 121. L'ambiguïté métaphorique des machines

Les interprètes sont divisés quant à l'importance du concept de *machine* dans l'émergence de la mécanique comme science centrale de la nature à l'époque moderne. Par exemple, pour E. J. Dijksterhuis, un des premiers historiens de l'histoire de la mécanique, seul le pôle mathématique de la mécanique fut significatif pour la révolution scientifique, et non son pôle « machinique ». Selon lui, le principal apport des machines cinématiques à la réflexion théorique fut de montrer que les phénomènes dynamiques étaient capables d'une investigation *mathématique*, c'est-à-dire non seulement qu'il était possible de déterminer avec précision les proportions entre les grandeurs caractéristiques des phénomènes, mais également d'en

126. (Roux 2013, p. 8).
127. (Popplow 2007, p. 65-68).
128. (Vérin 1993, ch. 1, § 2), et plus généralement (Dolza et Vérin 2004 ; Lefèvre, Renn et Schoepflin 2012).
129. Charles Sorel, *De la connaissance des bons livres* (1671), cité par (Roux 2013, p. 74).

avoir une connaissance *démonstrative*[130]. À ce titre, c'est la géométrie plutôt que l'algèbre qui resta le modèle de l'exposition mathématique de la science, jusqu'à Newton inclus, car c'est elle qui, dans la conception héritée des Grecs, donne la mesure de la démonstration certaine[131].

On peut également interpréter le rôle de la machine dans la réflexion des modernes comme celui d'une simple métaphore des phénomènes naturels. En faveur de cette hypothèse, on peut évoquer les hésitations de Boyle, qui le premier énonça explicitement les principes de la « philosophie mécanique », quant à l'appellation de son programme scientifique : il hésite entre « mécanique », « corpusculaire » ou « phénicienne » (car les fondateurs de l'atomisme, avant Démocrite, auraient été phéniciens) mais il favorise finalement le premier terme simplement parce qu'il évoque bien les phénomènes du mouvement, qui sont « évidents et très puissants dans les engins mécaniques (*mechanical engines*)[132] ».

On pourrait donc tout au plus reconnaître à la machine le rôle d'une métaphore pédagogique, qui fonctionnerait à peu près ainsi : *de la même manière* que nous n'avons besoin que de grandeur, figure et mouvement, pour expliquer démonstrativement le comportement d'un automate, *il n'y a pas besoin, de la même manière,* de supposer d'autres raisons pour expliquer tel ou tel phénomène naturel.

Contre cette hypothèse, on peut d'abord argumenter qu'il n'y a pas de lien historique ni conceptuel entre la métaphore de la machine et le développement de la science mécanique de la nature. Il suffit pour cela de considérer le cas de l'astronomie et de l'image de la *machina mundi*.

Cette image commence en effet à être associée aux horloges à la fin du XIV^e^ siècle – la première occurrence se trouve chez Nicolas Oresme, vers 1380[133]. L'association du mouvement des cieux à celui d'une machine se développe en effet avec l'apparition des horloges astronomiques à cette époque, qui voit également les astronomes chercher à tester leurs hypothèses astronomiques dans des modèles d'automates[134]. Dans tous ces développements, il n'est pas encore question d'une « mécanique céleste », au sens où l'entendront plus tard Kepler, Galilée et Descartes. Ainsi, pour Giovanni Dondi, qui conçut et fabriqua l'automate le plus

130. (E. J. Dijksterhuis 1986, p. 498).
131. (ibid., p. 50-54). Roux (2017, p. 31-32) donne d'autres exemples d'interprétations minimisant la pertinence des explications « machiniques » de la nature.
132. (ibid., p. 27), (Garber 2013, p. 7).
133. (Popplow 2007, p. 65).
134. Voir l'étude de (Shank 2007) à ce sujet.

sophistiqué de son époque pour représenter le mouvement des planètes, l'astrarium (1364), celui-ci sert à visualiser le système ptolémaïque et à en montrer la possibilité mathématique, mais nullement à rechercher ses causes[135]. L'automate à cette époque sert bien plutôt le camp des causes finales, car sa régularité montre la matière soumise aux lois de la raison, ou encore la raison *réalisée* dans la matière, sans les accidents et les imperfections congénitales aux formes naturelles. Ainsi Marsile Ficin, l'un des premiers à identifier l'expression *machina mundi* à celle d'un automate, exprimant sa fascination pour une machine de fêtes démontrée à Florence en 1475, en fait un argument en faveur du finalisme néo-platonicien, contre le mécanisme atomiste de Lucrèce :

> Nous avons vu récemment à Florence un petit cabinet fabriqué par un artisan allemand dans lequel des statues de différents animaux étaient toutes reliées à une seule boule et maintenues en équilibre par celle-ci. Quand la boule bougeait, ils bougeaient aussi, mais de façon différente : les uns couraient à droite, les autres à gauche, vers le haut ou vers le bas, les uns assis se levaient, les autres debout tombaient, les uns couronnaient les autres, et ceux-ci en blessaient d'autres. On entendait aussi le bruit des trompettes et des cors et le chant des oiseaux ; et d'autres choses se passaient là simultanément et une foule d'événements semblables se produisaient, et cela simplement à partir du mouvement d'une balle. Ainsi Dieu par son propre être, qui est en réalité le même que son entendement et sa volonté, ou qui est quelque chose d'entièrement simple – le centre universel à partir duquel (comme nous l'avons déclaré ailleurs) le reste des choses provient comme des lignes – n'a qu'à hocher la tête et tout ce qui dépend de lui tremble.
>
> N'écoutons plus Lucrèce l'épicurien, qui veut que le monde naisse et soit porté par le hasard, et qui croit que la condition constante de son ordre, beau et plein de formes, procède d'une privation instable et informe d'ordre [...][136]

Ici la merveille, qui est donnée comme analogie de la Création divine, n'est pas tant que la matière se meuve d'elle-même, car elle le fait bien par hasard[137], mais qu'elle se meuve conformément à la prévision de l'esprit[138].

La machine n'est donc pas du tout perçue comme une récusation des causes finales par les aristotéliciens ou les néo-platoniciens, bien au contraire. Aristote avait clairement distingué les causes finales naturelles et nécessaires, qui étaient principes de mouvement, des causes finales contingentes, c'est-à-dire les intentions

135. (ibid., p. 19-21).
136. *Théologie Platonicienne* (1482), éd. Hankins et Bowen, 2001, p. 201.
137. Voir plus haut, § 108.
138. (Crombie 1994, vol. 1, p. 468).

humaines, qui devaient user des causes efficientes pour les faire advenir[139]. Dans un cadre chrétien, les causes finales nécessaires deviennent elles-mêmes l'œuvre de l'intellect divin, comme on l'a vu plus haut chez Thomas d'Aquin. Mais il n'est nullement impliqué par là qu'il doit les concevoir par composition de causes efficientes. Il s'agit certes là du mode d'effectivité des causes finales humaines, mais en Dieu le mouvement est entièrement subordonné à la raison :

> Car il n'a pas affaire à des matériaux étrangers qu'il a reçus d'ailleurs, mais à ses propres matériaux qu'il fabrique lui-même. Il ne les affecte pas de l'extérieur, mais les fait bouger de l'intérieur. Il est présent au cœur même de toutes choses. […] Parce que les différentes parties du monde, nées de semences particulières et dotées de formes distinctes, cherchent et cherchent encore des objectifs spécifiques – les cherchent par la voie la plus directe, aux moments et dans les dispositions appropriés, et d'une manière à la fois très belle et idéalement adaptée – il s'ensuit qu'elles sont mues de la même manière que ces choses mises en œuvre par l'habileté et la conception humaines[140].

Il est donc important de noter que rien, dans le concept propre de *machine* ne permet de récuser la notion de cause finale. Au contraire : en évoquant l'ingénieur qui la conçoit, elle pointe vers l'idée d'une cause intentionnelle. Si les atomistes tenteront de l'éliminer par un raisonnement à la limite (telle machine est produite par la coopération fortuite de machines plus simples, elles-mêmes produites de la même façon, etc.), un raisonnement inverse est également possible, comme on le voit dans le texte de Marsile Ficin, où la structure grossière de la machine fabriquée par l'homme tend, à la limite de son perfectionnement, vers une osmose entre l'intention divine et la structure matérielle.

Marsile Ficin utilise donc à rebours (pour nos yeux contemporains) la métaphore de la machine, en n'en faisant plus l'emblème du triomphe des causes efficientes, mais au contraire en montrant comment la comprendre de manière organique, comme un lieu où la matière s'ajuste parfaitement à la raison, dans une image qui peut d'ailleurs rappeler le texte de Séris sur l'adéquation de la machine industrielle à la forme exacte du geste nécessaire[141], et plus généralement la conception organique de la machine prônée par Simondon.

139. Voir plus haut, § 108.
140. *Théologie Platonicienne*, II.13, (Ficin [1482] 2001, p. 203).
141. Voir plus haut, § 86.

§ 122. L'interprétation mathématique

Dijksterhuis et Koyré attribuent à la machine un rôle supérieur à celui d'une simple métaphore, en ce qu'elle montrerait la possibilité de mathématiser les phénomènes du mouvement. Pour Koyré, les accouplements de roues d'engrenage, qui servent à la transmission de la force motrice dans les appareils de levage et les moulins, sont des moyens « qui, positivement, *invitent* au calcul[142] ».

Koyré affirme que c'est l'application de l'intelligence théorique à la technique – soit la technologie – qui permit de répondre systématiquement à cette « invitation » que font les machines aux mathématiques :

> [Cette] possibilité trouve son expression et sa garantie dans le fait même que l'acte de l'intelligence qui, en décomposant et recomposant une machine, en *comprend* l'agencement, ainsi que la structure et le fonctionnement de ses rouages multiples, est exactement analogue à celui par lequel, en décomposant une équation en ses facteurs, elle en comprend la structure et la composition[143].

Selon Koyré, les prouesses techniques du Moyen Âge, aussi impressionnantes qu'elles furent, se contentèrent de méthodes par essais et erreurs et de l'utilisation de recettes éprouvées, car les inventeurs de cette époque *n'avaient tout simplement pas l'idée* qu'on put chercher la précision et la certitude démonstrative dans le monde environnant, ce monde « sublunaire » qui imposait à toute chose un écart, une approximation, des accidents relativement à son essence. Si l'astronomie et l'optique étaient mathématisables, c'est qu'elles portaient sur des phénomènes considérés alors comme immatériels, donc susceptibles de précision et de certitude. Ce qui permit donc à la technique de devenir technologie et de s'engager dans la voie du progrès indéfini fut, d'après Koyré, une idée proprement philosophique, que la spéculation (mathématique) pouvait s'appliquer aussi bien au monde sublunaire qu'à l'étude du ciel.

En particulier, c'est l'idée qu'eut Galilée de concevoir des instruments scientifiques – le télescope et le chronomètre – pour observer avec précision les phénomènes de la nature qui montra qu'une autre attitude épistémique était possible dans la conception d'artefacts :

> Pour faire les appareils optiques il faut non seulement améliorer la qualité des verres que l'on emploie et en déterminer — c'est-à-dire mesurer d'abord et calculer ensuite — les angles de réfraction, il faut encore améliorer leur taille, c'est-à-dire savoir leur

142. (Koyré 1961, p. 316).
143. (ibid., p. 316).

> donner une forme précise, une forme géométrique exactement définie ; et pour ce faire, il faut construire des machines de plus en plus précises, des machines mathématiques qui, autant que les instruments eux-mêmes, présupposent la substitution, dans l'esprit de leurs inventeurs, de l'univers de précision au monde de l'à-peu-près[144].

De la même manière, selon Koyré, ce n'est pas des progrès de l'horlogerie que provint le chronomètre, c'est-à-dire un *instrument* de mesure exacte du temps, mais de la *déduction rationelle* par Galilée de l'isosynchronisme du pendule, qui permit de l'utiliser comme mesure du temps et d'en faire par la suite la nouvelle base technique de la conception des montres[145].

Cette explication n'est pas entièrement satisfaisante pour plusieurs raisons. D'abord, des deux exemples que prend Koyré (le téléscope et le pendule), le premier concerne l'optique plutôt que la mécanique, et le second est une machine simple, où il n'est nullement question de roues et d'engrenages qui « inviteraient au calcul ». Il s'agit d'ailleurs, comme le dit Koyré, d'instruments plutôt que de machines. Par ailleurs, de nombreuses explications par les machines, notamment celles concernant les phénomènes organiques et complexes, sont très souvent qualitatives. Elles convoquent les propriétés structurelles des machines pour expliquer les phénomènes plutôt que des calculs.

Ensuite, la généalogie que fait Koyré de la technologie moderne, qui la subordonne à l'« esprit théorique » et à une nouvelle conception du monde, est aujourd'hui contestée. Comme nous l'avons dit plus haut, les philosophes modernes se targuent souvent du temps passé dans les ateliers à observer les artisans. Roberts, Schaffer et Dear (2007) ont consacré tout un ouvrage à critiquer cette représentation de la séparation faite entre savants et artisans dans des entreprises communes de découverte et d'innovation qui, d'après eux, est souvent motivée par des considérations sociales.

Si une distinction doit être faite elle passe, semble-t-il, entre les projets de machines dont la conception préalable est rigoureuse et justifiée géométriquement, et les projets de « mécaniciens » plus commerçants qu'ingénieurs, comme on peut le sentir dans cet avertissement de Mersenne dans sa préface à sa traduction des *Mécaniques* de Galilée :

> C'est pourquoi j'ajoute seulement qu'avant qu'on entreprenne les ouvrages où les Machines doivent entrer, et que l'on se serve des ingénieurs et artisans, qu'il est à

144. (Koyré 1961, p. 322).
145. (ibid., p. 328).

> propos de leur faire exposer leurs desseins et leurs modèles en public, et particulièrement à la vue des excellents Géomètres qui savent les vraies raisons de toutes sortes de Machines, et qui peuvent prévoir les inconvénients et les obstacles de l'air, de l'eau, et des autres circonstances, à faute de quoi il arrive trop souvent que plusieurs font des dépenses excessives, etc.[146]

L'approche mathématique de problèmes techniques n'est nullement une nouveauté, comme nous l'avons rappelé plus haut, et elle n'est pas encore nettement associée à l'« esprit théorique » évoqué par Koyré. F. J. Dijksterhuis[147] rapporte comment le père d'Huyghens s'était senti offensé de voir son fils qualifié de *mathématicien* dans une correspondance : ce terme était associé en effet, à l'époque, aux mathématiques pratiques des ingénieurs, et seul le titre de *géomètre* était socialement respectable[148]. De la même manière, Descartes note, dans le *Discours de la Méthode*, comment son éducation l'avait initialement porté à voir les mathématiques comme « seulement utiles aux arts mécaniques[149] ». Les mathématiques ont à cette époque un statut mixte entre théorie et pratique, et il semble donc bien qu'elles fussent utilisées à bon escient dans les projets techniques où il importait de prendre des décisions préalables précises et rigoureusement justifiées.

La nouveauté n'est donc pas d'utiliser les mathématiques dans la conception d'artefacts, mais de transposer ces méthodes à l'étude des phénomènes naturels. Il s'agit là de la thèse centrale que défend l'historien des sciences Nicholas Jardine (1976), en replaçant les innovations méthodologiques de Galilée dans le contexte de la philosophie naturelle de son époque. Deux conclusions de son analyse sont particulièrement pertinentes pour notre propos.

D'abord, les écrits méthodologiques du XVI^e^ siècle, notamment ceux de Nifo et de Zabarella, avaient déjà décrit l'investigation scientifique comme la conjonction de l'induction des phénomènes de l'expérience vers leurs causes et la démonstration réciproque de leur nécessité à partir de celles-ci. On a ainsi voulu y voir des prémisses de l'approche expérimentale de la science moderne. Ce que Nifo et Zabarella ne parvenaient pas cependant à expliquer de manière satisfaisante est qu'il n'y avait pas de cercle logique dans cette démarche, et comment on pouvait s'assurer, sinon par une certaine « intuition intellectuelle » que les premiers principes

146. (Galilei [1634] 1966, p. 19).
147. Un homonyme d'Eduard Jan Dijksterhuis mentionné ci-dessus.
148. (F. J. Dijksterhuis 2007, p. 59).
149. Édition d'Adam et Tannery, t. VIII (par la suite abrégé en A.T. VIII), p. 8.

postulés étaient valides. La rupture de Galilée est d'affirmer que la seule manière d'établir des démonstrations sur des faits naturels est d'abandonner le raisonnement syllogistique et de les considérer géométriquement, c'est-à-dire à l'aide de la seule discipline qui dispose de principes premiers absolument assurés[150].

Ensuite, ce que Galilée montre de son usage de la géométrie, c'est qu'elle conjoint *effectivement* deux fonctions que lui avaient attribuée les Anciens, sans pour autant voir leur unité : d'une part, elle peut être utilisée pour résoudre des problèmes pratiques dans l'expérience concrète, – comme Aristote le perçoit quand il compare la délibération pratique à une investigation géométrique, et d'autre part elle permet également de dégager les principes et axiomes sous-jacents des sciences et de la métaphysique – comme dans la tradition platonicienne[151]. Jardine peut ainsi conclure :

> Chez Galilée, la double association de l'analyse [géométrique] est également évidente, car son « chemin vers la vérité » est présenté à la fois comme une technique permettant de résoudre des problèmes spécifiques et comme un moyen d'appréhender des principes évidents. Ces associations de l'analyse géométrique avec le raisonnement pratique d'une part, et avec l'ascension vers les premiers principes d'autre part, étaient certes des lieux communs de la Renaissance. Mais cela [...] ne doit pas nous faire oublier l'éclat et la profonde originalité de l'utilisation par Galilée de l'analyse géométrique comme métaphore d'une forme de recherche scientifique qui fusionne en une seule activité la résolution de problèmes pratiques et la quête de principes théoriques[152].

La question qu'il faut donc se poser, si on veut tout de même faire droit à l'interprétation de Koyré qu'il existe un lien intime entre machines et mathématisation, est de savoir *pourquoi* les machines, et notamment les machines cinématiques, furent le lieu privilégié aussi bien du passage du savoir pratique approximatif au savoir exact, que de l'importation d'une posture technique dans l'approche scientifique, c'est-à-dire d'une démarche constructive dans l'étude des phénomènes.

7.5 Machine et constructibilité

Cette section, qui continue la précédente, examine d'abord le rôle que les modernes firent jouer aux machines dans leur entreprise scientifique, à savoir celui

150. (Jardine 1976, p. 305-310).
151. (ibid., p. 316-317).
152. (ibid., p. 317).

de concepts explicatifs des phénomènes (§ 123). Nous rapprochons ce privilège épistémique de leur caractère *constructible*, au sens géométrique – qui explique également leur importance pour la mathématisation des phénomènes de la nature, évoquée à la section précédente (§ 124). Enfin, nous avançons qu'un tel schéma reste toujours actif dans l'explication scientifique actuelle. Le déplacement décisif est que, si on ne demande plus à l'explication de *construire* les machines sous-jacentes au phénomène en question, on requiert à présent de spécifier quelles lois, déjà établies par la théorie, régulent leur comportement. L'objet de l'explication n'est plus une machine, mais un *système* dont le comportement, régi par des lois théoriques, suffit pour rendre compte des phénomènes. Ce déplacement de l'idéal de l'explication, de la *constructibilité* vers l'*instrumentalité* des entités posées par l'investigation scientifique, n'est pas une rupture qui sépare les pré- et les post-newtoniens, mais au contraire la maturation d'un même projet de connaissance (§ 125).

§ 123. L'interprétation mécaniste

Comme tout artefact[153], la machine est définie d'une part par sa fonction et de l'autre par sa structure dynamique. Sophie Roux, que nous allons suivre dans ces développements, précise simplement que la structure d'une machine doit être « suffisamment composée » pour que l'artefact puisse prendre cette dénomination.

Selon elle, c'est justement cette dualité qui, pour de nombreux philosophes des Temps modernes, fait de la machine, bien plus qu'une métaphore, un modèle d'intelligibilité des phénomènes, qui donne lieu à ce qu'elle appelle des « explications machiniques »[154]. Descartes, bien sûr, est l'un des grands initiateurs de ce genre d'explications : dans ses traités éponymes, il reconstruit le monde et l'être humain comme des machines fonctionnant seulement par « grandeur, figure et mouvement ».

Ce type d'explications est très critiqué depuis Newton[155]. Au mieux, elles permettent de construire un *modèle* du phénomène à expliquer, qui peut être résolu mathématiquement et vérifié dans l'expérience – comme par exemple l'usage par

153. (Kroes 2012).
154. (Roux 2013, p. 71).
155. Voir un résumé des positions critiques à ce sujet (Roux 2017, p. 31-33). Dans ce paragraphe, nous suivons la classification des explications mécanistes de (Gabbey 2001) et les études détaillées de cas de (Roux 2013). Nos exemples sont tirés du premier article.

Descartes du rebond d'une balle de paume pour modéliser les lois optiques de la réflexion et de la réfraction. Dans la plupart des cas, cependant, les explications se contentent de proposer des analogies entre le phénomène à expliquer et des relations structurelles observées dans le vie quotidienne – qui restent cependant elles-mêmes non expliquées. Par exemple, Boyle compare le « ressort de l'air » (par quoi il entend la force de dilatation que l'air exerce après avoir été comprimé) à celle d'une pelote de laine, composée de petits cheveux qui ont chacun une certaine élasticité. On peut ici parler d'une « esquisse de mécanisme » dont la modélisation selon des lois connues de la matière et du mouvement est cependant encore lointaine. Enfin, dans le pire des cas, on reste dans le domaine de la métaphore abstraite qui n'explique rien, ou de la simple supposition de figures ou de propriétés des éléments, qui seraient producteurs du phénomène considéré, ce qui rappelle l'explication circulaire de l'effet de l'opium par sa vertu dormitive. Par exemple, on expliquera l'odorat par la figure spécifique des molécules olfactives qui ne peuvent entrer en relation qu'avec celles de l'organe olfactif. Ces excès explicatifs sont ainsi commentés par l'historien des sciences Alan Gabbey :

> En un sens, il semble que les mécanistes se soient auto-infligé leurs difficultés à expliquer les sensations. Ils essayèrent de tout expliquer, ce qui était bien au-delà de leurs moyens, et leur propre critique des modes d'explication traditionnels ne les alerta pas des dangers qu'il y avait à vouloir remplacer le péripatétisme en bloc. La physique des formes et des qualités était taillée sur mesure pour tout expliquer : la philosophie mécanique ne l'était pas. L'efficacité explicative ne devait pas résider dans l'exhaustivité, mais dans une délimitation sélective du domaine du problème […] L'ontologie et les idéaux explicatifs des mécanistes étaient nouveaux, mais très souvent le « domaine du problème » était largement péripatéticien[156].

De manière plus positive, on peut rapprocher l'*intention* des philosophes modernes d'un mode d'explication scientifique légitime, qu'on appela un temps *structurelle*[157], avant que le terme *mécaniste* reprenne le dessus, avec l'émergence du mouvement des « nouveaux mécanistes » au début des années 2000. Ce mouvement vise à faire droit à l'explication structurelle dynamique des phénomènes en sciences de la vie d'abord, mais dans d'autres sciences aussi en général[158]. À la différence des explications « machiniques » des philosophes modernes cependant, les parties constitutives du mécanisme explicatif ainsi que leurs relations causales

156. (Gabbey 1984, p. 14).
157. (McMullin 1978).
158. Voir à ce sujet plus loin notre discussion section 11.5.

ne sont pas élaborées par analogie ou par métaphore : elle sont fondées sur des théories scientifiques établies par ailleurs. Ainsi, l'explication d'un certain comportement d'une cellule par certaines actions des mitochondries, du noyau, de l'appareil de Golgi, etc. n'est pas du ressort de l'analogie, ni même du modèle : il s'agit de la résolution d'un phénomène global en composantes comportementales élémentaires elles-mêmes déjà établies ou observées. Lorsque des hypothèses doivent être faites concernant un élément ou une action encore mal connus, elles doivent pouvoir être en droit explorées et falsifiées par l'expérimentation.

Si on se place du point de vue des *intentions* des philosophes modernes, il s'agit là d'une différence de pertinence plus que d'essence. Les explications d'aujourd'hui sont simplement bien mieux dotées en instruments explicatifs que celles d'il y a quatre cents ans. Sophie Roux met en avant deux points communs essentiels entre elles : d'abord, ces explications se situent à « à mi-échelle » : décrire une machine ou un mécanisme n'exige pas de le résoudre en ses composants les plus élémentaires et en lois premières du mouvement. Les composants et les mouvements que l'explication manipule peuvent être pris comme des points d'étape dans l'analyse du phénomène et de sa réduction à des lois élémentaires de la nature. Comme nous venons de le dire, les nouveaux mécanistes admettent l'idée d'hypothèses et donc d'*esquisse de mécanismes* – ce que furent également les explications des philosophes modernes, au moins pour les meilleures d'entre elles.

Le second point commun de ces explications est de s'adresser d'abord à la *compréhension* de l'investigateur. Elles visent à dévoiler la structure interne du phénomène en question, ainsi que les relations causales entre les parties de cette structure, causant ensemble à leur tour le phénomène global. Ce dévoilement ne passe pas naturellement par la proposition et le syllogisme démonstratif, mais plutôt par des diagrammes qui permettent de les appréhender plus intuitivement, notamment par la simulation mentale de leur fonctionnement. Ici, Sophie Roux rappelle que Descartes se serait inspiré, pour expliquer le fonctionnement du monde ou des organes humains, de la forme d'explications illustrées qu'on pouvait trouver dans les théâtres de machines, et qu'il fut le premier à inclure des schémas dans des ouvrages de philosophie naturelle[159].

Cette compréhension qui se constitue dans les diagrammes de machines nous semble faire écho à l'évocation par Koyré[160] de « l'acte de l'intelligence qui, en

159. (Roux 2017, p. 36-41).
160. Voir plus haut, p. 479.

décomposant et recomposant une machine, en *comprend* l'agencement, ainsi que la structure et le fonctionnement de ses rouages multiples ». Les machines invitent en effet à une compréhension *active* qui met en rapport leur fonctionnement avec leur structure d'une part, et celle-ci avec les actes de sa construction.

Or c'est bien la possibilité de cette *connaissance active* des phénomènes naturels qui nous semble centrale pour les modernes, et qui fonde leur mathématisation possible. Cela est très visible chez le premier Descartes, et notamment dans sa lettre à Froidmont de 1637. Froidmont s'était plaint que la philosophie de Descartes, ressemblant à celle d'Épicure, était « crasse et mécanique », à quoi Descartes, visiblement piqué, reprit l'adjectif à son compte :

> Si ma philosophie lui parait trop grossière, parce qu'elle ne considère que des grandeurs, des figures et des mouvements, comme fait la Mécanique, il condamne en elle ce que j'estime qu'on doit louer par-dessus tout, et dont je suis particulièrement fier et glorieux : à savoir, que j'use d'une façon de philosopher telle, que tous mes raisonnements ont une évidence mathématique, et que des expériences véritables confirment toutes mes conclusions ; si bien que tout ce que j'ai conclu qui est possible en vertu de mes principes, on le réalise effectivement, chaque fois, bien entendu, qu'on applique ce qui est actif à ce qui est passif (*adeo ut quicquid ex ejus principiis fieri posse concludi, fiat revera, quoties activa passivis, ut par est applicantur*). Je m'étonne qu'il n'ait pas fait lui-même cette remarque : La Mécanique qui a été jusqu'ici en usage, n'est qu'un chapitre de la vraie Physique, et rien d'autre ; mais, comme la Philosophie vulgaire n'a pas su lui garder sa place, elle s'est retirée chez les Mathématiciens. Or cette partie de la Philosophie est demeurée plus vraie et s'est moins gâtée que les autres : comme elle se rapporte à l'usage et à la pratique, les erreurs qu'on y commet se paient d'ordinaire par une perte d'argent : si bien que, s'il méprise ma manière de philosopher, parce qu'elle ressemble à la Mécanique, c'est, semble-t-il, comme s'il la méprisait d'être vraie[161].

Selon Sophie Roux, cette lettre marque un pivot dans la conception cartésienne de la mécanique[162]. Jusqu'ici, Descartes avait opposé, en géométrie, les procédés « exacts » par lesquels on construisait des courbes aux procédés « mécaniques » se servant d'instruments complexes[163]. Il associait donc au terme à peu près la même connotation dépréciative qu'entend Froidmont, évoquant un critère d'efficacité pragmatique indigne de la vraie science. Dans sa réponse, Descartes considère au

161. *Lettre du 3 octobre 1637 à Froidmont*, A.T. I, p. 420, traduction dans l'édition de la *Correspondance* par Adam et Milhaud (1939), t. 2, p. 16.
162. (Roux 2004, p. 31-33).
163. Voir à ce sujet les citations données par Bos (2001, p. 6).

contraire la mécanique dans son statut de mathématique mixte, qui l'élève au rang des savoirs certains. Mais il fait sien également le sens entendu par Froidmont en répliquant ironiquement que l'ignorance de cette science « se paie par une perte d'argent » – « *se paie cash* », dirait-on aujourd'hui en langage familier.

En d'autres termes, c'est une même chose pour une machine d'être démontrable mathématiquement, réalisable et effective. Ce que Descartes entend par « application de l'actif au passif » dans la phrase centrale n'est pas entièrement clair. Elle est interprétée généralement[164] comme pointant vers l'action de l'esprit qui réalise ses idées dans les corps appropriés, c'est-à-dire vers l'action technique. Le sens général de l'argument va en tous cas dans cette direction : tout ce qu'on peut conclure des principes comme réalisable *peut l'être effectivement*.

§ 124. La constructibilité des machines

L'association du caractère réalisable des machines à leur caractère mathématique tient à deux traditions anciennes. D'une part, il y a la tradition de la science mécanique, remontant aux Grecs, qui affirme qu'on peut construire toutes les machines complexes à partir des cinq machines simples – ainsi que Descartes lui-même l'écrit dans une lettre contemporaine de celle à Froidmont, qui expose les machines simples :

> Et il serait utile pour ceux qui se mêlent d'inventer de nouvelles machines, qu'ils ne sussent rien de plus de cette matière que ce peu que je viens d'écrire ; car ils ne seraient pas en danger de se tromper en leur compte, comme ils font souvent en supposant d'autres principes[165].

D'autre part, la constructibilité des machines simples tient elle-même à ce qu'elles sont des figures géométriques dans lesquelles les règles élémentaires du mouvement peuvent être le mieux aperçues. Or la géométrie est au XVII^e^ siècle le savoir certain et exact par excellence, par le fait que le géomètre sait *construire* ses objets à partir des seuls instruments autorisés par les postulats d'Euclide, la règle et le compas. Bos (2001) a montré la centralité de cette notion de *construction* dans les recherches des géomètres jusqu'au moins 1650, concernant les instruments additionnels *admissibles* pour obtenir les objets qui résistaient à toute construction par la règle et le compas, et le rôle central que joue Descartes

164. Voir la note de F. Alquié à ce propos dans son édition (t. 1, p. 792, note 1), ainsi que Grene (1995, p. 203).

165. *Lettre à Huyghens du 5 octobre 1637*, A.T. I, p. 447.

dans ces recherches[166]. La question qui se posait aux géomètres, selon Bos, était de déterminer quels objets géométriques pouvaient être dits *connus* exactement, et la marque de cette connaissance n'était pas encore – comme de nos jours – de posséder l'équation de la figure, mais de pouvoir la construire. La trisection de l'angle, la quadrature du cercle, le tracé de certaines courbes comme la spirale sont des exemples de problèmes qu'on ne savait pas résoudre avec la règle et le compas, mais avec des instruments additionnels, comme des compas mobiles[167]. Descartes va certes introduire l'approche algébrique de la géométrie ; cependant elle est encore pour lui un moyen de résoudre les problèmes géométriques, c'est-à-dire de construire des figures[168].

La solution de Descartes au problème de l'exactitude va être d'admettre comme géométriques toutes les courbes qui peuvent être tracées d'un seul mouvement continu et régulier, ce qui inclut celles qui sont constructibles par le compas articulé qu'il avait inventé[169], mais exclut par exemple la spirale, qui requiert deux mouvements simultanés. Selon Descartes, il s'agit là d'une simple condition d'intelligibilité ; réciproquement, rien n'est plus intelligible que ce mouvement de construction, qu'il place au fondement de la géométrie dans cette remarque du *Monde* :

> La nature du mouvement duquel j'entends ici parler est si facile à connaître que les géomètres mêmes qui, entre tous les hommes, se sont le plus étudiés à concevoir bien distinctement les choses qu'ils ont considérées, l'ont jugée plus simple et plus intelligible que celle de leurs superficies et de leurs lignes ; ainsi qu'il paraît en ce qu'ils ont expliqué la ligne par le mouvement d'un point et la superficie par celui d'une ligne[170].

C'est de ce mouvement même que Dieu anime l'étendue qu'il a divisé en parties de figures et grandeurs différentes[171].

La constructibilité et la réalisabilité ne sont donc pas ici des propriétés empiriques des machines, qu'on observerait en décomposant, par exemple, une machine complexe en pièces d'assemblage, ou en conjecturant, comme l'ingénieur moderne, une certaine résistance des matériaux, certaines propriétés chimiques, etc. Il s'agit d'une constructibilité *a priori*, qui a pour propriété, non seulement de réaliser la chose prévue, mais également de permettre la démonstration de toute

166. Voir également (Blåsjö 2022) sur ce sujet.
167. (Bos 2001, p. 6).
168. (ibid., p. 226).
169. (ibid., p. 237).
170. *Le Monde*, ch. VII, A.T. XI, p. 39.
171. *Le Monde*, ch. VI, A.T. XI, p. 36. Nous nous inspirons ici de (Domski 2009).

sorte de théorèmes à son sujet – parce que ses principes constitutifs sont entièrement clairs pour l'esprit, n'étant eux-mêmes que des opérations de l'esprit.

Cette position que le vrai savoir, qui trouve sa perfection dans la géométrie, réside dans la capacité du sujet à concevoir son objet constructivement, n'est pas propre à Descartes. L'historien de la philosophie Popkin l'a nommée « scepticisme constructif » et, pour l'historien des sciences A.C. Crombie, elle correspond au style de raisonnement scientifique qu'il appelle *modélisation hypothétique*[172]. Elle remonterait, selon lui, à la Renaissance et à la tradition du *Timée*, qui avait permis à Thomas d'affirmer que la connaissance divine devait être conçue sous la modalité pratique[173]. Appliquer l'idée de machine à l'étude des phénomènes naturels, c'est promouvoir l'analyse géométrique de leur structure (ainsi l'œil est conçu sur le modèle de la *camera obscura*) et de leurs processus (ainsi les mouvements célestes sont-ils conçus sur le modèle de l'horlogerie)[174]. Cette position, qu'on ne connaît vraiment que ce qu'on sait construire, est exprimée notamment par Mersenne, Gassendi, et Hobbes. Nous ne pouvons faire l'historiographie complexe de cette position[175], et il nous suffit de citer deux phrases de Mersenne qui en sont caractéristiques :

> Il est difficile de rencontrer des principes, ou des vérités dans la Physique, dont l'objet appartenant aux choses que Dieu a créées, il ne faut pas s'étonner si nous n'en pouvons trouver les vraies raisons, et la manière dont elles agissent, et pâtissent, puisque nous ne savons les vraies raisons que des choses que nous pouvons faire de la main, ou de l'esprit[176].
>
> L'homme n'est pas capable de savoir la raison d'autre chose que de ce qu'il peut faire, ni d'autres sciences, que de celles, dont il fait lui-même les principes, comme l'on peut démontrer en considérant les mathématiques[177].

On peut noter, dans la première citation, l'alternative *ce que nous pouvons faire de la main, ou de l'esprit*, qui pose comme au fond équivalentes les constructions géométriques et mécaniques.

Sans projeter la position de Mersenne à l'ensemble des modernes, elle nous permet de dégager une interprétation du concept de *machine* qui nous sera utile

172. Voir plus haut, § 111.
173. Voir plus haut, § 111.
174. (Vaccari 2008, p. 288-290).
175. On peut se référer ici à (Crombie 1994) et à (Pérez-Ramos 1988).
176. *Harmonie Universelle*, Nouvelles observations physiques et mathématiques, 1636, t. 2, p. 8.
177. *Questions Théologiques*, 1634, q. 22, p. 111

dans la suite de cette réflexion, à savoir qu'étant un objet *construit* géométriquement, elle dispose *par là* de propriétés démontrables. Or ces propriétés ne sont pas, comme dans le cas du triangle par exemple, la somme de ses angles ou l'intersection de ses hauteurs, mais tout simplement celles de son *mouvement*. Par l'intermédiaire des machines, on peut espérer dégager des démonstrations géométriques des phénomènes dynamiques, et donc les expliquer et les prévoir avec la certitude des mathématiques. La machine est l'objet de la *géométrie du mouvement*.

§ 125. La résolution scientifique de problèmes

La notion de constructibilité permet de voir la solidarité des deux précédentes interprétations du mécanisme. D'une part, la machine a joué un rôle pivot dans la mathématisation de la nature, parce que sa constructibilité doit être entendue ici en un sens géométrique, celle-là même qui permet à cette discipline d'atteindre la pleine certitude. D'autre part, elle a favorisé les explications structurelles car sa structure a ceci d'attractif qu'elle est totalement analysable *en droit*, également du fait de sa constructibilité.

Cette position va être remise en cause par l'avènement de la physique newtonienne et son célèbre *Hypotheses Non Fingo* (« je ne feins pas d'hypothèses »), qui vise directement les explications « machiniques », notamment cartésiennes. Ce déplacement épistémologique a déjà fait l'objet de nombreuses analyses et interprétations[178]. Nous nous intéressons seulement ici à son implication pour les concepts de *constructibilité* et de *machine*.

Ce sur quoi Newton en particulier ne fait pas d'hypothèses, c'est sur la *cause* de la gravitation universelle, ce qui implique qu'on doit la voir comme un *fait* ou un *phénomène* qu'on peut mesurer et dont on peut établir inductivement une théorie mathématique, sans que celle-ci dise quoi que ce soit des raisons de ce comportement.

Cette position a été longtemps interprétée dans le sens du positivisme logique. Pour Ernst Mach, Newton aurait compris que l'objet réel de la science était de donner une description mathématique des régularités phénoménales de la nature, où le concept métaphysique de *cause* n'avait nulle place[179].

178. Nous avons notamment consulté : (Brading 1999 ; Cohen 1969 ; Gabbey 1992, 1996 ; Levitin 2021 ; McMullin 2001 ; Walsh 2012).
179. (Mach [1883] 1919, p. 193, 246).

Dans une telle vision, qui fait de l'établissement et de l'application des lois de la nature le contenu essentiel de la science, toute explication machinique peut paraître conjecturale ou au mieux métaphorique, servant peut-être seulement à mettre l'investigation sur la voie d'une bonne formulation ou d'une bonne application de ces lois.

C'est contre une telle vision que s'élève McMullin (1978), dans un article célèbre qui anticipe le mouvement des « nouveaux mécanistes » en philosophie des sciences[180]. McMullin oppose ce qu'il appelle l'*explication structurelle* à l'*explication nomothétique* promue par les empiristes logiques, et déclare la première préférable à la seconde. Ainsi, expliquer l'élongation de la barre de fer sous l'effet de la chaleur est plus satisfaisant lorsqu'on exhibe le comportement de sa structure moléculaire que lorsqu'on se contente d'évoquer la loi de dilatation des métaux. Cette loi empirique ne peut en effet être établie que sur la base d'un raisonnement du premier type. McMullin va ainsi remettre en cause l'interprétation positiviste de Newton et tenter de montrer que sa physique propose bien des explications structurelles[181].

Cependant cette opposition nous semble artificielle. Les deux formes d'explication, nomothétique et structurelle, sont compatibles et complémentaires, tout en se situant à des niveaux de discours différents. On peut résoudre un problème en faisant seulement appel à la loi de dilatation des métaux parce que celle-ci a déjà été établie dans un corpus théorique. Mais cela veut dire qu'elle a été dérivée d'autres lois plus élémentaires par exemple concernant les effets de l'augmentation de l'énergie cinétique d'une structure rigide et, comme le dit McMullin, une telle dérivation passe par une explication structurelle. Toute résolution de problème est en partie structurelle, et c'est sa partie constructive, et en partie nomothétique, et c'est là sa partie descriptive, puisqu'il faut bien décrire les relations et les objets qui sont à la base de cette interaction structurelle.

Une explication peut paraître seulement nomothétique lorsque le problème explicatif posé se trouve être très proche d'une question typique qui a déjà été intégrée au corpus théorique pour des raisons de commodité ou d'illustration. C'est le cas de la loi de dilatation des métaux, un phénomène courant et important. Ainsi, si on demande pourquoi une barre de fer s'allonge lorsqu'elle est chauffée, il suffit de reconnaître qu'il s'agit là d'une question déjà traitée. Cette loi ne peut cependant

180. Voir ci-dessus § 123, et plus loin, section 11.5.
181. (McMullin 2001).

fournir une explication satisfaisante que parce qu'on dispose de sa dérivation à partir des lois fondamentales de la science. Tant qu'elle n'était qu'une loi empirique qu'on ne savait pas encore intégrer au corpus systématique de la science, les explications qui la mobilisaient étaient seulement provisionnelles.

L'opposition que croit voir McMullin entre explications structurelles et nomothétiques prend donc son origine dans celle qui existe entre deux formes d'investigation, la résolution de problèmes particuliers et la construction d'un corpus théorique systématique. La résolution de problèmes permet d'établir de nouvelles lois scientifiques qui s'agrègent aux théories, mais celles-ci forment à leur tour le socle de toute résolution de problème.

Le *hypotheses non fingo* de Newton interdit les explications structurelles seulement à propos des principes *élémentaires* du système de la science, mais il les développe au contraire dans tout le reste de son corpus : la loi de l'attraction universelle fut jugée admirable par le nombre de telles explications qu'elle permit de dériver, concernant la vitesse de la chute des corps, les marées, le mouvement des planètes, etc.

Aussi – et c'est le point principal auquel nous souhaitons arriver – les machines et leur construction ne sont nullement bannies de la physique newtonienne ; c'est seulement de la « machinerie » des lois élémentaires de la nature dont il est affirmé qu'il ne faut rien conjecturer. On peut, d'une certaine manière, interpréter cela comme une tautologie qui ne fait que reprendre l'assertion aristotélicienne selon laquelle les principes d'une science ne peuvent être démontrés : car si on pouvait donner une explication structurelle de ces lois, alors elles ne seraient plus élémentaires. C'est bien ce que fit d'ailleurs Einstein lorsqu'il expliqua la loi de la gravitation comme un simple effet de la loi d'inertie dans un espace courbé par des masses corporelles.

L'explication scientifique nous semble ainsi toujours procéder par la mise en évidence de comportements structurels globaux à partir des lois de comportement de leurs parties. Cette mise en évidence peut passer par des explications mécanistes, qui sont majoritaires en sciences de la vie, où les « machines » sont les organismes, les tissus, les cellules ou leurs composantes, à la fois objets et moyens de l'explication. En sciences physiques la palette d'explications est plus large. Elle passe par exemple souvent par des considérations statistiques. Sans entrer ici dans des considérations épistémologiques complexes à ce sujet, nous remarquons seulement que là encore reste centrale l'idée d'un emboîtement de comportements structurels,

qu'on sait mettre en relation selon des règles précises. C'est par de telles explications que peuvent être articulés l'un à l'autre différents niveaux descriptifs des phénomènes que proposent (par exemple) la chimie, la thermodynamique, la physique atomique, la physique nucléaire, et que leurs lois respectives peuvent espérer être intégrées (au moins idéalement) en un système unifié du savoir.

C'est ce fait élémentaire de l'explication scientifique qui se situe dans la continuité des « explications machiniques » tentées par les modernes, à savoir que, pour *appliquer* une loi, il faut bien circonscrire l'objet ou le groupe d'objets auxquels on veut l'appliquer, et donc distinguer un *système d'intérêt* d'un environnement avec lequel il interagit. Si par exemple on veut appliquer la loi de conservation de l'énergie ou la seconde loi newtonienne de la dynamique, il faut bien identifier les corps dont on va calculer l'energie, la masse, le bilan des forces et l'accelération, pour mettre en évidence le phénomène cible de l'explication. *Système*, ici, est un terme méthodologique employé par les scientifiques pour désigner *ce dont ils parlent*, et dont ils veulent démontrer les effets. Chaque théorie décrit, implicitement ou non, la manière dont les systèmes peuvent y être représentés, ainsi que Vorms le montre à propos des versions newtonienne et lagrangienne de la mécanique[182]. C'est là que se situe la continuité d'une telle méthode avec celle que promeut Descartes dans ses traités physiques.

La différence essentielle se situe ailleurs. Alors que les modernes se contentent souvent d'analogies tirées de la vie courante pour décrire leurs mécanismes, l'explication scientifique depuis Newton exige qu'on explicite quelle loi démontrée ou provisionnelle précise on *utilise* pour décrire le comportement du système d'intérêt ou de ses composantes, et de laquelle l'explication va être dérivée. La focalisation de l'explication n'est plus sur la construction de machines, mais sur leur usage. Quand Newton explique les phénomènes des marées, la Terre, la Lune, les masses d'eau se comportent bien comme des machines, mais on n'exige plus de comprendre *pourquoi* il en est ainsi. Il suffit de montrer que leur interaction produit le phénomène à expliquer.

Nous avons dit plus haut que la construction et l'utilisation d'une machine étaient deux mouvements articulés en miroir, puisque la construction de l'une pouvait se faire par l'utilisation d'une autre. Le déplacement essentiel que fait Newton est d'affirmer qu'il est vain (voire absurde) de vouloir reconstruire *toutes* les machines de la nature : il faut bien en poser certaines comme élémentaires, et

182. (Vorms 2009, p. 38, 51).

se contenter de connaître seulement les règles de leur *usage*. À partir d'elles, on peut alors entreprendre de reconstruire toute la machinerie du monde.

De là provient la couleur instrumentale qu'on a souvent associée à la manière newtonienne de faire de la science. Si l'explication d'un phénomène passe par l'utilisation d'une règle donnée décrivant le comportement interactif de machines sous-jacentes, plutôt que par la tentative de construire celles-ci de zéro et d'établir cette règle chaque fois à neuf, alors on peut également utiliser cette même règle à d'autres fins, comme *prédire* le phénomène en question, et dans certains cas en prendre techniquement le *contrôle* pour le mettre au service de nos desseins – puisqu'une disposition différente des machines sous-jacentes au phénomène engendrera de nouvelles interactions entre elles, dont on pourra dériver les conséquences par la même règle. Comme le dit Peter Dear :

> L'univers selon une philosophie naturelle comme celle de Newton, qui consistait en une matière sans vie rebondissant selon des lois mécaniques, était clairement exploitable à des fins humaines : il se prêtait à l'instrumentalisation[183].

L'instrumentalité n'est en effet que le miroir de la constructibilité. Elle en constitue une évolution et un « progrès », si on peut s'exprimer ainsi, en ce qu'elle acte la division du travail que la machine rend possible, entre sa construction et son usage, et leur relation récursive et cumulative, puisqu'on peut, par l'usage de machines, en construire de nouvelles[184]. De la même manière, la division du travail entre investigation systématique et résolution de problème permet, par l'usage des lois déjà établies par la science, d'en dériver de nouvelles.

7.6 Systèmes et théories

Nous achevons ce chapitre par l'examen de la notion de *système*. Celle-ci est déjà apparue à de nombreuses reprises lors de cette étude, d'abord dans un sens qui la rapproche de celle de *méthode*[185], puis dans deux sens « objectifs » (c'est-à-dire où système désigne un ensemble d'objets), l'un visant les systèmes techniques[186], et l'autre les systèmes physiques, objets de l'investigation scientifique[187]. Après un bref rappel de l'histoire de la réapparition et du succès de ce terme aux Temps

183. (Dear 2006, p. 10).
184. Voir plus haut, § 119.
185. Voir plus haut, § 113.
186. Voir plus haut, § 118.
187. Voir plus haut, § 125.

modernes (§ 126), nous étudions les fortunes d'un de ses sens aujourd'hui oublié, qui fut pourtant son sens principal dans la première moitié du XVIII^e siècle en France, celui d'*hypothèse*. Cette notion épistémologique fut très vite critiquée, tant en science qu'en philosophie, par la philosophie expérimentale anglaise d'une part, par le mouvement éclectique allemand de l'autre (§ 127). Les hésitations terminologiques des encyclopédistes français montrent la difficulté de synthétiser ces deux critiques et finissent par amener le remplacement de *système* par *théorie* dans la désignation des doctrines scientifiques (§ 128).

Nous terminons ce chapitre par une courte synthèse de ses principaux résultats (§ 129).

§ 126. La notion de système aux Temps modernes

Le terme *σύστημα* est courant en grec ancien, et il désigne d'une manière générale toute sorte d'assemblages organisés, qu'il s'agisse d'institutions humaines (l'armée, un État), de choses naturelles ou de réflexions. Les Stoïciens jouent un rôle particulièrement important dans l'orientation du terme vers son sens moderne. D'une part, l'expression *système du monde* est née dans un contexte stoïcien :

> Le monde est le système du ciel, de la terre et des natures qui s'y trouvent.
> Κόσμος μὲν οὖν ἐστι σύστημα ἐξ οὐρανοῦ καὶ γῆς καὶ τῶν ἐν τούτοις περιεχομένων φύσεων[188].

D'autre part, comme on l'a vu plus haut, la technique (*τέχνη*) est définie comme un système de représentations. Victor Goldschmidt résume très bien le caractère intrinsèquement systématique de la philosophie stoïcienne, dans le cadre duquel la cohérence des citations précédentes peut être comprise :

> Plus que toute autre école antique, le stoïcisme a insisté sur le caractère systématique de sa philosophie. […] [Le système] se présente sous deux formes : comme une progression méthodique, d'une part, et d'autre part et plus profondément, comme une totalité organique. Sous le premier aspect, système signifie l'*ordre des raisons*. L'exposé des différentes thèses ne se fait pas n'importe comment, et même il est impossible de développer telle thèse séparée, détachée de l'ensemble. […] L'ordre des raisons est conforme à l'ordre des choses. Mais cela même nous avertit que cet ordre,

188. Pseudo-Aristote, *De Mundo*, 391b9. Le *de Mundo* est généralement considéré comme un traité d'origine stoïcienne. Cf. également un passage correspondant chez Diogène Laërce, qui attribue l'expression à Posidonius, *SVF 2.526*.

> fondé en être, recouvre un *système* plus profond qui, lui, n'est pas progression qui s'accomplit, mais totalité achevée. [...] L'ordre des raisons va être ramené au *Logos* total, la progression méthodique se concentrera et s'absorbera dans la sagesse parfaite qui doit être présente d'un coup, et dont l'achèvement instantané correspond à celle de l'univers ; ce n'est qu'en attendant d'y parvenir que l'apprenti-philosophe s'exerce dans ce qu'on pourrait appeler le temps méthodologique, et parcourt en sens différent, le réel, toujours le même, qu'il ne possède pas encore sous forme de totalité[189].

Étant donné que *σύστημα* fut traduit en Latin par divers termes signifiant *assemblage*, comme *compages*, *collectio*, le mot *systema* ne fut globalement pas utilisé par le Moyen Âge, sauf en son sens musical, qui désigne une certaine forme d'harmonie[190]. C'est seulement à la fin de la Renaissance, lorsqu'il devint acceptable d'introduire des mots grecs dans la pratique de la langue latine qu'il réapparut enfin. Les deux sens vus ci-dessus de *système* vont alors être réactivés, mais dans des contextes séparés.

Si, comme nous l'avons déjà vu, son sens épistémique va être réintroduit au XVI^e^ siècle par les milieux philippo-ramistes et être rapidement associé à celui de *méthode*, *système* se popularise également à la même époque par une voie entièrement parallèle, située dans le cadre de la discussion sur le *système du monde* proposé par Copernic. Ce n'est pas Copernic lui-même qui utilise l'expression, mais son disciple Rheticus dans son ouvrage de 1540 exposant des thèses de son maître. Ce livre n'eut pas beaucoup d'écho initialement, mais il fut régulièrement réimprimé et lu par les astronomes, si bien que le terme devient central dans la polémique des « systèmes du monde » qui se développe dans les années 1580-1590 : on parle alors du *système ptolémaïque*, du *système de Copernic*, etc.[191]

Cette dénomination va avoir une double postérité, qui suit les deux manières dans laquelle on peut l'entendre. D'une part, *système du monde* peut désigner les choses elles-mêmes en tant qu'elles sont organisées. Ici il s'agit du système solaire, au sens où nous l'entendons aujourd'hui. Par extension de ce sens, *système* va bientôt désigner d'autres ensembles *objectifs* organisés, comme par exemple le *système digestif* (cf. ci-dessous).

D'autre part, *système du monde* peut désigner la *conception* qu'ont différents astronomes de l'organisation et de l'ordre régissant le mouvement de la terre, du

189. (Goldschmidt [1953] 1989, p. 61-63).
190. (Hager et Strub 1995, p. 827), (Lerner 2005, p. 410).
191. Voir l'historique de l'expression *systema mundi* présenté par (ibid.).

soleil, de la lune et des planètes. À ce titre, il se met très rapidement en place un *modus vivendi* épistémologique de l'*hypothèse*, destiné à affirmer la coexistence possible de plusieurs conceptions cosmologiques – les partisans de l'Église, car ils ne peuvent nier que les mathématiques modernes donnent raison, en terme de simplicité et d'élégance, aux Modernes, et ces derniers, car ils veulent se protéger d'accusations d'hérésie. Aussi *système* devient-il très rapidement, dans son usage philosophique, synonyme d'*hypothèse*, englobant et éclipsant pour un temps, surtout en France, son sens méthodique issu du ramisme, *i.e.*, l'exposé ordonné d'un corpus de règles ou de propositions. Un *système* en effet ne désigne pas seulement l'hypothèse elle-même, mais également toutes les vérités qui en sont dérivées logiquement.

C'est cette définition que les dictionnaires français retiennent comme principale à la fin du XVII^e^ siècle, comme par exemple Furetière, qui brosse un large panorama de ses emplois (nous ne citons son entrée que partiellement) :

> SISTEME, ou SYSTEME, s. masc. Terme d'astronomie. Supposition, ou hypothèse que font les astronomes d'un certain ordre, ou d'un certain arrangement des parties de l'univers, sur le fondement de laquelle ils expliquent tous les phénomènes, ou apparences, qui se trouvent dans le cours des astres ou dans leurs changements. Il n'y a de différence entre système, et hypothèse, sinon que l'hypothèse est un système plus particulier, et le système, une hypothèse plus générale. Le système de Ptolémée, de Copernic, de Tyco Brahé, de Fracastor. Le Père Deschales prétend qu'on peut inventer jusqu'à vingt systèmes, ou hypothèses, qui expliquent avec une égale précision toutes les apparences des astres -, en regardant comme immobile quelques-uns des neuf termes que nous avons, c'est à savoir les sept planètes, la Terre, et le firmament.
>
> Ce mot vient du Grec, et signifie composition.
>
> On appelle aussi en physique le système des sens, du mouvement, de la nourriture, etc. la manière dont on suppose, et on conçoit que les organes sont disposés. Alors il signifie, constitution, situation.
>
> Entre les médecins il y en a qui suivent le système des saveurs ; d'autres qui suivent le système des quatre qualités ; d'autres le système des acides et des alcalis. Ce médecin fait un nouveau système des fièvres, c'est-à-dire, qu'il suppose de certains principes, suivant lesquels il explique toute la nature et les symptômes des fièvres. Les théologiens ont fait aussi divers systèmes de la Grâce. […]
>
> Ce mot s'emploie aussi au figuré. Le système des affaires de la Cour. […][192].

192. Dictionnaire de Furetière (1701). Orthographe modernisée.

Outre son sens d'*hypothèse* le sens « objectif » de *système* apparaît dans l'exemple des structures physiologiques, comme l'appareil digestif, d'où est dérivé également son sens politique ou institutionnel (« le système des affaires de la cour »), qui apparaît encore comme « figuré ». Il est intéressant de noter le soin que prend Furetière à distinguer les systèmes physiologiques des systèmes « des médecins » qui ne sont pas des structures organiques mais bien des hypothèses médicales – ce que nous appellerions des théories, comme le « système des fièvres ».

Ces deux sens principaux à la fin du XVII^e^ siècle en France vont avoir une fortune différente, mais également importante pour notre propos. L'emploi de *système* comme « structure objective » va se généraliser jusqu'à devenir omniprésent aujourd'hui, dans les sciences naturelles, les sciences humaines (où il désigne notamment des phénomènes institutionnels et sociaux) et les techniques. Cette signification conduit à la proposition de van Bertanlaffy, dès les années 1930, de créer une *théorie générale des systèmes*[193]. Nous réservons son analyse au dernier chapitre de notre réflexion.

Par contre, *système* comme *hypothèse* a complètement disparu de nos usages. Dans le développement suivant, nous retraçons l'histoire de cet « accident épistémologique », singulier aux Temps modernes, qui eut de fortes implications sur notre conception de la relation entre théorie et pratique et qui permit l'émergence de la notion de *théorie scientifique*.

§ 127. La critique des systèmes

Dans la définition de *système* par Furetière[194], il se trouve une allusion aux « systèmes de la Grâce ». Elle se réfère à la polémique concernant le « système des causes occasionnelles » de Malebranche, qui va diffuser le terme en métaphysique. On va parler alors, dès les années 1680, du système de Descartes ou de Spinoza, et Leibniz intitule lui-même sa philosophie le « système de l'harmonie préétablie ». On s'aperçoit que le caractère hypothétique des explications « machiniques » de Descartes a un « double fond » : il ne s'agit plus seulement de savoir si les explications mécanistes des phénomènes par « grandeur, figure et mouvement » sont correctes[195], mais *quel serait le sens même* à les dire telles, puisque les construc-

193. (Bertalanffy [1968] 2015).
194. Voir plus haut, p. 497.
195. Voir plus haut, § 123.

tions de l'esprit sont incommensurables avec les affections des corps.

Cette multiplicité des systèmes d'astronomie et de philosophie, que les échanges d'arguments ne semblent pas pouvoir départager, va favoriser la diffusion d'une posture sceptique et hostile à l'idée de système. Malebranche lui-même l'évoque dès la première édition de son traité *De la Recherche de la Vérité* en 1675, si bien qu'on peut dire qu'en philosophie, la critique de l'idée de système lui est congénitale. En particulier, deux postures tentent d'échapper au scepticisme. La plus ancienne est celle de la philosophie expérimentale anglaise, qui se forme dans les années 1660 autour de la *Royal Society*, et qui souhaite réguler l'usage des hypothèses en sciences. La seconde est l'éclectisme, qui se développe vers le début du XVIII^e^ siècle en Allemagne, et qui concerne plus proprement l'attitude philosophique. Les encyclopédistes français, que nous examinons au développement suivant, réalisent dans les années 1750 la synthèse malaisée de ces deux mouvements.

1) L'école empiriste anglaise hérite de Bacon sa méfiance à l'égard des constructions systématiques trop hâtives et fait l'éloge de la patience requise par la méthode expérimentale dans la collecte des faits, leur vérification et leur mise en relation. En particulier, les empiristes anglais préfèrent à *système* l'usage du terme *théorie* pour désigner leurs doctrines. Il s'agit là d'une particularité terminologique mineure, car cet usage reste rare et n'est jamais (à notre connaissance) l'objet d'une promotion explicite. Il sera cependant décisif pour son adoption, un siècle plus tard, pour désigner les doctrines scientifiques, et il est donc utile de nous y arrêter.

Jusqu'à la fin du XVI^e^ siècle l'adjectif *théorique* a une prééminence naturelle par rapport au substantif *théorie*, qui n'apparaît que rarement. Ses premières occurrences pour désigner des doctrines scientifiques utilisent d'ailleurs le terme *theorica* dérivé de sa forme adjectivale[196] : ce terme apparaît dans un emploi spécialisé au XV^e^ siècle pour désigner les traités d'astronomie[197], et parfois des horloges mécaniques dont l'astrarium s'inspire[198]. Le dictionnaire savant de Glocenius, paru en 1613, mentionne seulement l'adjectif *théorique*, qui se réfère à son sens ancien, spéculatif.

Nous n'avons pas trouvé d'histoire détaillée de la diffusion du terme *théorie* au XVII^e^ siècle. Il commence à y être employé régulièrement dans différents contextes

196. (O. Pedersen 1961).
197. (ibid., p. 161).
198. Sur l'astrarium, voir plus haut § 121.

qui dérivent de son usage astronomique mais qui ne sont pas nécessairement scientifiques. Son parcours est donc très largement équivalent à celui de *système*, avec l'avantage qu'il est moins associé aux querelles astronomiques, et qu'il reste ancré dans la conception mathématique et instrumentale de cette science, simplement descriptive des mouvements des planètes, et muette sur leurs causes. Il garde aussi, à ce titre, sa vieille connotation « contemplative ».

Boyle utilise *théorie* de manière interchangeable avec *hypothèse*. Contrairement à *système* cependant, qui selon Furetière est l'« hypothèse plus générale[199] », une bonne théorie est pour Boyle une hypothèse locale et faillible, qui s'accorde avec les observations passées, mais qui doit également soutenir le test des expériences futures :

> On peut attendre d'une bonne hypothèse qu'elle soit en accord non seulement avec toutes les autres vérités, mais aussi avec tous les autres phénomènes de la nature, ainsi qu'avec ceux qu'elle est censée expliquer. Ceci étant accordé (ce qui ne peut être nié), celui qui établit une *théorie* doit non seulement veiller à ce qu'aucun des phénomènes naturels déjà observés ne contredise son hypothèse pour le moment, mais aussi à ce qu'aucun phénomène découvert par la suite ne la contredise pour l'avenir[200]. *(Nous soulignons)*

Newton, quant à lui, utilise ce terme dès 1672 dans sa *Theory of Light and Colors*, un titre peut-être suggéré par Oldenburg. Cependant, dans sa correspondance, notamment avec Hooke, il l'utilise dans un sens bien distinct de Boyle, en opposant cette fois *théorie* à *hypothèse*. Hooke avait critiqué la doctrine de Newton en déclarant infondée son hypothèse corpusculaire de la lumière. À quoi celui-ci répond que Hooke fait ici une confusion logique. Il n'avance cette hypothèse qu'à la fin de son traité, comme une conjecture qu'on peut raisonnablement *inférer de sa théorie*, cependant celle-ci, seulement fondée sur des observations et sur leur description mathématique, n'en dépend aucunement. C'est donc bien parce qu'on a *déjà* une théorie qu'on gagne le droit à émettre des hypothèses. Cette position se fonde, à notre avis, sur une remarque qui clôt sa longue lettre, que la doctrine des couleurs devrait être considérée comme une mathématique mixte, dont les principes physiques sont réduits à des observations de fait entièrement certaines[201]. La distinction entre *théorie* et *hypothèse* est donc nette chez Newton : la première est du ressort des sciences mathématiques, et la seconde de la

199. Voir plus haut, § 126.
200. (Boyle 1674, p. 207).
201. *Lettre à Hooke*, 11 juin 1672 (Newton 1959, vol. 1, p. 187).

philosophie naturelle. En 1672, Newton importe le terme de l'astronomie à l'optique et, quelques années plus tard, c'est dans le même sens qu'il l'utilisera dans les *Principia*, quoique très discrètement, à propos de la mécanique. D'une manière générale, l'usage du terme *théorie* par les scientifiques anglais, quoique positif, est très rare, et il n'aura pas de postérité immédiate.

2) En philosophie, une manière d'échapper à la querelle des systèmes autrement que par le scepticisme va être l'éclectisme, un mouvement qui traverse le XVIII^e^ siècle mais qui naît d'abord en Allemagne au début du siècle avec les figures de Johann Christoph Sturm (1635 - 1703) et Christian Thomasius (1655 - 1728). L'éclectisme promeut la liberté de penser par soi-même. Il enjoint l'esprit éduqué de ne pas se laisser entraîner dans l'une ou l'autre « secte » philosophique, mais de *choisir* dans les divers systèmes les différentes propositions qui lui semblent le mieux s'accorder à la vérité. Christian Wolff critique cette idée dès 1729, dans son opuscule *De differentia intellectus systematici & non systematici*, en argumentant qu'un éclectisme désordonné ne peut être qu'un syncrétisme (Wolff n'utilise pas le mot, mais Diderot le fera) et que le vrai esprit éclectique doit également être systématique. C'est bien de cet esprit systématique que Wolff fait d'abord l'apologie, et des systèmes seulement par ricochet.

C'est le sens méthodique de *système* qui refait ici surface avec Wolff, celui qui fut popularisé en Allemagne par les ouvrages de Keckermann, Timpler et Alsted, et qui ne s'applique pas seulement à la métaphysique et à la philosophie naturelle, mais à toute discipline. Dans son traité, Wolff donne pour illustrer l'idée de *système* des exemples tirés des mathématiques, de l'astronomie, du droit, etc. Et c'est donc bien dans un esprit ramiste du « penser par soi-même » que Wolff peut affirmer que le vrai philosophe éclectique sera profondément systématique :

> Ceux qui possèdent une intelligence systématique sont libres du préjugé de l'autorité et sont capables de procéder en éclectiques. En effet, ils n'admettent que ce dont on peut prouver qu'il est contenu dans un système par ses principes. Ils jugent donc pour des raisons internes, mais non pour des raisons externes […] Celui qui jouit d'un esprit systématique survole rapidement les écrits d'autres auteurs. Il passe outre ce qui est déjà contenu dans son système et porte son attention uniquement sur les propositions qu'il ne connaît pas encore. S'il les comprend ou les explique selon ses propres termes, il s'efforce de les déduire du concept du sujet par des principes issus de son système. S'ils sont trouvés vrais par ces principes, il les ajoute à son système […] II en ressort combien se trompent ceux qui opposent la philosophie éclectique

à la philosophie démonstrative[202].

§ 128. L'esprit systématique et les théories scientifiques

Les encyclopédistes français sont les héritiers de ces deux traditions critiques qu'ils ont peine à concilier. D'une part, les éclectiques à la manière de Wolff voient dans la systématicité la condition même de la philosophie. D'autre part, le crédit qu'ils accordent à Newton leur fait condamner la constructions de systèmes comme contraire à l'esprit scientifique. Leur posture vis-à-vis de l'idée de systématicité est donc ambivalente, ce qui s'observe dans leurs hésitations terminologiques. Dans l'*Encyclopédie*, Diderot présente ainsi le penseur éclectique comme le vrai philosophe qui évite tout à la fois le sectarisme des systèmes, le scepticisme qui les rejette tous dans leur entièreté, et le syncrétisme qui les mélange sans méthode[203]. Par contre, dans la plupart des articles de l'*Encyclopédie*, l'expression *esprit systématique* est le plus souvent péjorative, comme ici dans l'article *Philosophie* qui lui oppose le « véritable esprit philosophique », dont la description rejoint celle de l'éclectique :

> L'esprit systématique ne nuit pas moins au progrès de la vérité : par esprit systématique, je n'entends pas celui qui lie les vérités entre elles, pour former des démonstrations, ce qui n'est autre chose que le véritable esprit philosophique, mais je désigne celui qui bâtit des plans, et forme des systèmes de l'univers, auxquels il veut ensuite ajuster, de gré ou de force, les phénomènes[204].

Dans cette assertion, le terme *système* est si négativement connoté que Diderot tente de couper ses liens profonds avec les notions d'*ordre* de la connaissance et de *méthode*, en proposant de qualifier ce que Wolff appelait *systématique* de *philosophique* tout simplement.

Une telle proposition avait peu de chance d'aboutir, et c'est plutôt celle de d'Alembert que la postérité retiendra lorsque, dans un passage célèbre du *Discours Préliminaire à l'Encyclopédie* (1751), il utilise *systématique* dans le sens opposé à celui de Diderot, et dénomme plutôt *esprit de système* ce que ce dernier signifie par là :

> Ce n'est donc point par des hypothèses vagues et arbitraires que nous pouvons espérer de connaître la Nature ; c'est par l'étude réfléchie des phénomènes, par la

202. (Wolff [1729] 2019, p. 76).
203. Article *Éclectisme* de l'*Encyclopédie*, vol. 5, p. 271a. Voir (Carboncini 1987) sur l'influence discrète de Wolff sur l'*Encyclopédie*.
204. *Encyclopédie*, article « Système », vol. 15, p. 515a

> comparaison que nous ferons des uns avec les autres, par l'art de réduire, autant qu'il sera possible, un grand nombre de phénomènes à un seul qui puisse en être regardé comme le principe. Cette réduction, qui les rend d'ailleurs plus faciles à saisir, constitue le véritable esprit systématique qu'il faut bien se garder de prendre pour l'esprit de système, avec lequel il ne se rencontre pas toujours[205].

Il faut d'ailleurs noter qu'ici l'opposition entre *esprit systématique* et *esprit de système* n'est pas franche, puisque d'Alembert se contente de remarquer que les deux « ne se rencontrent pas toujours », ce qui laisse sous-entendre qu'ils le font parfois, ou même souvent. Plus loin il appelle l'esprit de système *esprit d'hypothèse et de conjecture* en se référant positivement aux hypothèses locales et faillibles de Boyle[206]. Celles-ci ont bien un rôle positif à jouer en sciences, et il faut seulement éviter qu'on les confonde avec des « conjectures frivoles » et qu'elles deviennent des principes inébranlables, afin que le physicien n'en fasse jamais des « systèmes ».

C'est donc bien du terme *système* qu'il faut entièrement se débarrasser afin que les idées qu'elle véhicule – celle de méthode et celle d'hypothèse scientifique – puissent regagner du crédit. Et c'est bien dans le *Discours Préliminaire* qu'on trouve, peut-être pour une des toutes premières fois, la suggestion de remplacer le terme *système* par celui de *théorie*, à propos de Newton :

> Sa Théorie du monde (car je ne veux pas dire son Système) est aujourd'hui si généralement reçue, etc.[207]

Peut-être d'Alembert emprunte-t-il le mot *théorie* à un autre auteur, mais il est en tous cas contemporain des années 1750. Voltaire, par exemple, ne l'utilise pas encore dans ses *Éléments de la philosophie de Newton* de 1737, quoiqu'il sonne la charge contre les systèmes. Le *Traité des systèmes* (1749) de Condillac, si virulent également à leur encontre, désigne tout de même, faute de mieux, la physique de Newton comme un *système* dont les principes sont des phénomènes[208], une expression étrange, qu'il oppose aux « systèmes abstraits » des philosophes aux « systèmes de suppositions » des scientifiques cartésiens.

La petite phrase de d'Alembert est en tous cas témoin de cette substitution encore mal assurée. Elle va se généraliser dans le dernier quart du XVIII^e^ siècle. Un siècle plus tard, elle ne pose aucun problème à Claude Bernard, qui se rappelle encore très bien l'ancienne signification de *système*, quoiqu'il l'attribue faussement

205. *Encyclopédie*, vol. I, page vi.
206. *Encyclopédie*, vol. I, page xxxi.
207. *Encyclopédie*, vol. I, page xxvi.
208. (Condillac [1749] 2010, p. 187).

à la « scolastique » :

> En méthode expérimentale, l'hypothèse est une idée scientifique qu'il s'agit de livrer à l'expérience. L'invention scientifique réside dans la création d'une hypothèse heureuse et féconde ; elle est donnée par le sentiment ou par le génie même du savant qui l'a créée. Quand l'hypothèse est soumise à la méthode expérimentale, elle devient une théorie ; tandis que, si elle est soumise à la logique seule, elle devient un système. Le système est donc une hypothèse à laquelle on a ramené logiquement les faits à l'aide du raisonnement, mais sans une vérification critique expérimentale. La théorie est l'hypothèse vérifiée, après qu'elle a été soumise au contrôle du raisonnement et de la critique expérimentale. […] En un mot, les systèmes et les doctrines en médecine sont des idées hypothétiques ou théoriques transformées en principes immuables. Cette manière de procéder appartient essentiellement à la scolastique et elle diffère radicalement de la méthode expérimentale[209].

ll faut remarquer que c'est le sens boylien de *théorie empirique*, et non newtonien de *théorie mathématique* que retient ici Claude Bernard. Leur coexistence jusqu'à aujourd'hui et leur compatibilité est une question de philosophie des sciences qui ne nous intéresse pas ici.

En remplaçant *système*, *théorie* va se charger de nombreuses de ses connotations. Dans le langage courant, on parle encore d'une théorie comme d'une hypothèse qui reste à vérifier. Le terme va se diffuser hors du domaine scientifique et désigner également les ouvrages de *réduction en art*, – on écrit ainsi des *Théories des jardins*, mais le terme se généralise au-delà encore, et on voit ainsi apparaître également à cette époque des *théories du bonheur* par exemple[210].

Théorie garde également son sens ancien, qui l'associe à la contemplation des principes. Une théorie n'est pas seulement une « machinerie » efficace pour répondre à des questions concernant la nature et pour résoudre des problèmes. Elle permet également de rendre plus évidents quels pourraient être les principes des phénomènes, en fournissant les démonstrations qui relient les uns aux autres – dans la démarche essentielle de *réduction* des définitions que décrivait déjà Aristote dans *APo.* II.8[211]. Ce que nous appelons aujourd'hui *recherche fondamentale* porte cette ambition spéculative. Il n'est pas du ressort de cette réflexion de discuter quel peut être l'espoir ou même le sens profond de cette ambition, qui a opposé

209. (Bernard [1865] 2013, p. 372).
210. (König et Pulte 1998, p. 1136). On peut consulter, par exemple, *la Théorie des Jardins*, Paris, 1776, de Jean-Marie Morel.
211. Voir plus haut, § 105.

dans les trente dernières années réalistes et pragmatistes (entre autres). L'important pour nous est de bien reconnaître cette double visée de l'entreprise scientifique que Galilée, selon Nicholas Jardine cité plus haut[212], avait réussi à si bien articuler grâce à l'emploi avisé de la géométrie : progresser dans la compréhension des principes du réel *et* résoudre des problèmes relatifs aux phénomènes. L'historien des sciences Peter Dear soutient une thèse proche de celle-ci : pour lui, la science moderne émerge, à partir de Galilée, comme un « amalgame de philosophie naturelle et d'instrumentalité » :

> Dans la culture moderne, la « science » peut être représentée sous sa forme d'une philosophie naturelle ou sous sa forme d'instrumentalité, mais pas sous les deux à la fois. Lorsqu'un énoncé scientifique est considéré comme relevant de la philosophie naturelle, il a le statut d'une description du monde naturel. On peut éventuellement en faire quelque chose, mais en l'état, il a simplement trait au monde tel qu'il est. Inversement, lorsqu'un énoncé scientifique est considéré comme l'expression d'une instrumentalité, il s'agit d'un rapport sur la manière de faire quelque chose, dont on peut également dire qu'il a des implications sur la manière d'être du monde. Les énoncés scientifiques sont intrinsèquement ambigus en ce qui concerne le type d'énoncé qu'ils revendiquent, naturel-philosophique ou instrumental[213].

§ 129. Synthèse

Il est utile de récapituler ici les principaux enseignements de ce chapitre. À la Renaissance, suite aux transformations de l'idéal et du concept de connaissance qui ont lieu dès le Moyen Âge, l'ambition systématique qu'Aristote réservait aux sciences s'étend à tous les domaines du savoir. Cette systématicité est d'abord conçue comme *pédagogique*, c'est-à-dire comme un dispositif extérieur visant à éduquer l'esprit à acquérir une systématicité intérieure, autrement dit à le rendre capable de s'orienter dans la connaissance par lui-même, et d'augmenter ainsi son pouvoir d'agir dans le monde.

Cette ambition est cependant confrontée à un problème formel : car de nombreux domaines de connaissance ne pouvant de manière évidente exhiber de principes, c'est une approche « cartographique » du savoir qui est privilégiée, dont l'indexation en tables et en arborescences est la forme la plus emblématique. Une telle démarche montre très vite ses limites, qui à notre avis ne sont pas techniques

212. Voir plus haut, § 122.
213. (Dear 2006, p. 8).

mais conceptuelles. Car les *règles* qui permettent à l'investigation de s'orienter dans la matière du savoir se révèlent ambivalentes, tantôt visant à *augmenter* la réflexion dans ses jugements, tantôt visant à la diriger exactement, voire à la rendre inutile.

Ce sont ces difficultés que le modèle *géométrique* de systématisation permet de résoudre, car la géométrie ne prétend pas donner de solutions de problèmes particuliers (sinon à titre d'exemples paradigmatiques) mais de fournir les concepts essentiels à la description et à la résolution rigoureuse de tout problème dans son champ. En géométrie, le système n'a pas pour fonction d'énumérer par une quelconque combinatoire ou arborescence tous les problèmes possibles, dont il donnerait immédiatement la solution – mais seulement de construire un espace ouvert pour la formulation indéterminée de tout problème et pour leur investigation efficace.

La géométrie permet de *démontrer* les propriétés des objets dont elle montre la *construction* possible, et en cela elle est l'écho de la connaissance parfaite par laquelle Dieu connaît les choses *en tant qu'il les a créées*. C'est ce double rapport construction / démonstration qui est ici essentiel. De manière décisive pour la technique moderne, à partir du XVII^e^ siècle l'ingénierie militaire va mobiliser la géométrie pour mettre en œuvre une nouvelle organisation des projets de construction, impliquant la normalisation de leurs opérations et l'instauration d'une triple division de travail, entre corps de métiers d'abord, entre maître d'ouvrage, ingénieur et contractant ensuite, et enfin entre savant et ingénieur. Dans ces divisions du travail se font jour de manière claire deux découplages essentiels pour la technique moderne : d'une part, celui entre la conception et l'exécution de l'ouvrage, et d'autre part, celui entre l'investigation systématique (globale) et la résolution de problèmes (locale).

La machine cinématique joue un rôle très spécifique dans ce mouvement. Se substituant à l'artisan, elle permet de rendre le découplage entre la conception et l'exécution exacte, transférant à la première le reliquat de jugement qui persistait dans la seconde. Du côté de l'exécution, les règles doivent devenir absolument déterminantes ; du côté de la conception, elles peuvent seulement guider l'investigation dans la résolution d'un problème qui doit à présent prendre la charge de tous les détails de l'exécution et prévoir toutes les configurations possibles de la situation.

Pour les mêmes raisons, la machine cinématique peut également devenir le

modèle de l'explication scientifique des phénomènes dynamiques. Si en effet expliquer le mouvement revient à démontrer les propriétés d'une structure qu'on sait construire, il devient envisageable de conférer la certitude de la géométrie à notre compréhension du monde. Il nous a semblé à cet égard que la rupture newtonienne relativement aux explications « machiniques » des cartésiens ne marquait pas un abandon de ce modèle épistémologique, mais sa maturation : car on s'aperçoit qu'il n'est pas nécessaire de réduire *toute* la physique à de la géométrie pour obtenir une explication satisfaisante d'un phénomène, mais que des théories qui décrivent les lois de comportement de leurs machines élémentaires (comme les corps newtoniens soumis à la gravitation) suffisent pour construire les machines complexes qui rendent comptent des phénomènes. Ce n'est qu'à propos de ces machines élémentaires qu'on abandonne toute prétention à les savoir construire, ce que traduit le *« hypotheses non fingo »*. Il faut bien noter ici que *construire* ne doit pas être entendu en un sens matériel, mais comme l'agencement selon des règles autorisées d'objets *postulés* par l'esprit, au sens euclidien de ce terme. En ce sens, l'explication statistique de phénomènes à partir des lois de comportement de micro-machines reste tout à fait conforme à ce modèle.

Avec la catégorie de la *machine*, il devient possible d'envisager une approche unifiée de la résolution de problème, qui couvrirait aussi bien les questions théoriques que les finalités pratiques. D'abord, l'intelligence que donne le savant à l'ingénieur de la machinerie du monde permet à celui-ci d'intervenir de manière toujours plus précise en son sein. Le point clé est ici que le savant s'exprime dans le langage même de l'ingénieur, celui de constructions que celui-ci peut donc immédiatement mobiliser : d'où l'association courante de la physique moderne à l'instrumentalisme. Mais là n'est pas l'essentiel, car ce fait lui-même résulte de l'adoption d'une démarche nouvelle et structurante de l'investigation, inspirée de la géométrie. Il s'agit d'abord d'accepter d'envisager tout mouvement ou toute action comme composé de mouvements et d'actions plus élémentaires dont on connaît les lois ou les règles ; et ensuite de chercher quelle peut être cette composition qui explique, ou produit le mouvement ou l'action désirés. Le soin de fournir ces objets élémentaires est confié à l'investigation systématique ; et celui de trouver les bonnes compositions dans chaque cas particulier, à la résolution de problème. L'idée devient donc possible de considérer que tout problème, pour être résolu, doit être plongé dans un espace de problème rigoureusement et exhaustivement constitué ; et que la charge de fournir ces espaces de problèmes incombe à

l'investigation scientifique.

Les tribulations du terme *système* aux Temps modernes sont à cet égard emblématiques de ces transformations. Il est d'abord associé aux méthodes des réductions en art, puis aux hypothèses physiques (la querelle copernicienne) et métaphysiques (la querelle des causes occasionnelles) du Grand Siècle, et enfin, dans sa substitution par *théorie*, à la démarche scientifique contemporaine. Nos théories visent d'une part à fournir un cadre systématique à la résolution de problème, et en ce sens le terme n'est pas réservé à la science, et peut également couvrir ce qui autrefois relevait de la réduction en art. D'autre part, nos théories, dans leur entreprise de réduction systématique, peuvent pointer également vers les principes du réel, quoique de manière problématique. Dans cette dernière acception, la conception aristotélicienne de la science est retrouvée.

Chapitre 8

Les règles du jugement et de l'action (Kant)

§ 130. Introduction

Nous avons exposé au chapitre 6 notre interprétation de la doctrine aristotélicienne du savoir technique, et montré en quoi le Stagirite pouvait le déclarer génériquement distinct du savoir théorique. Alors que le second vise à saisir les principes des choses, le premier relève simplement de la capacité à généraliser des délibérations.

La force apparente de cette position nous a amené à nous interroger sur les raisons qui permettent à Kant de la contester. Il nous a semblé que le rapprochement de la théorie et de la pratique opéré par Kant *venait de plus loin*, c'est-à-dire de déplacements conceptuels dont Kant lui-même était l'héritier, et qu'il ne faisait que mener à leur terme. Dans le texte de l'Introduction à la *Critique de la Faculté de Juger* déjà cité[1], il ne semble pas en effet énoncer une découverte philosophique majeure, mais plutôt appeler ses contemporains à la mise en cohérence de leur conception des champs du savoir avec des principes déjà communément admis.

À la fin du XVIII^e^ siècle, le regain de popularité du terme *théorie* est récent, et dû principalement à sa disponibilité pour se substituer au terme *système*, trop associé aux constructions métaphysiques du Grand Siècle et à l'attitude dogmatique dénoncée par d'Alembert et Condillac[2]. Il hérite cependant du *système* sa large portée sémantique, qui peut qualifier toute exposition ordonnée d'un domaine

1. Voir plus haut, p. 380.
2. Voir plus haut section 7.6.

phénoménal, qu'il s'agisse d'êtres naturels, de métiers ou de pratiques humaines. Cette malléabilité est justifiée par l'idée, souvent implicite, d'une raison et une méthode commune à tous les arts et les sciences, sans distinction. Ainsi chez Kant la *théorie* n'est pas encore aussi étroitement associée qu'aujourd'hui à l'idée de savoir scientifique, mais désigne plus généralement toute réflexion rationnelle à visée *systématique*. Là résiderait le socle commun de la connaissance des choses de la nature et des préceptes technico-pratiques, dans cette tendance de la pensée à s'organiser, qu'il s'agisse de refléter l'organisation sous-jacente du monde, ou de dégager les conditions d'efficacité de l'action.

Lorsque Kant rapatrie donc la technique dans le domaine de la connaissance théorique, il ne faut pas comprendre par là que ses préceptes sont dérivés des lois de la nature, au sens où la technologie a été conçue au XIX^e^ siècle comme la simple application de la science[3]. Il n'est pas nécessaire, pour être kantien sur cette question, d'adopter le réductionnisme naïf que nous avons critiqué lors de notre examen d'Aristote. Kant, comme on le verra, conçoit tout à fait qu'il existe des systèmes de connaissances indépendants empiriquement au sein même de la connaissance théorique. Leur unité ultime en un seul système de connaissance n'est qu'une idée régulatrice de la raison. L'appartenance commune des règles techniques et des lois naturelles à ce même système ne tient donc pas au fait que les unes dérivent des autres mais au fait qu'elles procèdent de la même faculté de connaître « par concepts naturels », et que Kant appelle l'entendement. Ce n'est pas sur des systèmes particuliers et empiriques qu'est fondée l'unité de la connaissance théorique et technique, comme s'il fallait s'assurer que toute règle technique pût être ramenée à une science déterminée, mais bien plutôt sur le fait que toutes deux répondent à la même exigence transcendantale de la raison, de ramener la diversité des règles de l'entendement à l'unité.

Ce chapitre est consacré à l'étude de cette position kantienne et à sa mise en débat avec celle d'Aristote. Kant parvient à achever les tendances profondes retracées au chapitre précédent, de *nier* qu'il y ait une distinction générique entre savoir théorique ((de *ce qui est*) et savoir pratique (de *ce qu'il faut faire*), en ce qu'il les ramène tous deux à l'idée de *règle* comme unité élémentaire de la connaissance discursive. La technique énonce des règles pour l'action qui ne sont pas essentiellement différentes de celles que l'entendement applique dans ses jugements (sec-

3. (Schatzberg 2018, p. 55 et suivantes). Voir aussi (Deldicque 2022) pour la critique de l'*applicationisme* par Canguilhem, Guillerme et Sebestik.

tion 8.1).

La position kantienne étant exposée, nous revenons aux trois distinctions entre théorie et pratique que notre analyse d'Aristote avait dégagées, et dont Thomas d'Aquin avait, dès le XIII^e siècle, déjà fortement contesté l'importance. Dans les sections 8.2 et 8.3 nous étudions donc successivement la pertinence de diviser la connaissance en théorie et pratique selon la contingence de ses objets (naturels ou intentionnels), selon sa modalité (qu'elle porte sur des faits ou des méthodes) et selon sa finalité (désintéressée ou « utilitaire »). À chaque fois nous donnons la parole à des philosophes contemporains pour défendre la position d'Aristote. Nous trouvons ces arguments convaincants, cependant, comme on le verra, nous concluons tout de même chaque fois à l'unité de la connaissance, en utilisant un même schéma qui prend ces arguments à contre-pied : car la question n'est plus, après les Temps modernes, de justifier l'autonomie du savoir pratique à l'égard de la théorie, mais de savoir si au contraire cette dernière peut être fondée sur des critères autres que pragmatiques ou instrumentaux.

La réflexion kantienne met néanmoins au jour une autre articulation de la connaissance, qui passe près de la distinction entre théorie et pratique sans s'y ajuster exactement : car s'il y a d'une part les investigations *locales* de l'entendement, qui rend l'expérience signifiante par l'application et l'élaboration de règles, il y a également les investigations (que nous appellerons *globales*) de la raison qui visent à organiser ces règles en *systèmes* de connaissances. C'est dans la mobilisation de ces systèmes, sans doute, que se situe la particularité de cette manière *systématique* de résoudre des problèmes que nous cherchons à élucider. Cependant, nous constatons que Kant se situe étrangement en retrait sur cette question qui avait grandement occupé Aristote (section 8.4).

8.1 Les règles du jugement et de l'action

Cette section est consacrée à l'examen des raisons qui permettent à Kant d'affirmer l'unité de la connaissance théorique et de la connaissance technique. Nous commençons par examiner la notion de règle, qui est centrale aussi bien dans la seconde (§ 131) que dans la première (§ 132). Nous étudions ensuite les formes du jugement telles que les présente la *Critique de la raison pure* et utilisons l'interprétation de LONGUENESSE (1993) afin de montrer leur accord profond avec la logique technicienne que nous avons commencé à esquisser dans les chapitres précédents

(§ 133). De la même manière, nous exposons l'idée kantienne du raisonnement comme *jugement complexe* afin de suggérer un premier rapprochement, dans le domaine technique, avec notre idée moderne de procédure (§ 134).

§ 131. Les règles de la technique

Pour exposer l'unité profonde de la connaissance théorique et technique dans une perspective kantienne, il est utile de repartir de la proposition centrale de Thomas d'Aquin, qui affirme la possibilité de considérer spéculativement le savoir pratique. Cela se produit, dit Thomas, lorsque l'architecte se demande : « *Si je voulais construire la maison, comment devrais-je m'y prendre ?* ». Pour qu'ait lieu la conversion du regard pratique en regard théorique, il faut donc *suspendre* l'intérêt pratique, il faut que la finalité recherchée devienne, en quelque sorte, hypothétique. Alors se révèle le caractère de *règle* du savoir pratique : « *Si tu veux A, alors fais B* », et le savoir spéculatif consiste à étudier ces règles en tant que telles.

Kant est donc en conformité avec cette tradition aristotélicienne et thomiste lorsqu'il affirme que le savoir technique porte sur des *règles* :

> Toutes les sciences ont une partie pratique, consistant en des problèmes qui supposent que quelque fin est possible pour nous, et en des impératifs qui énoncent comment cette fin peut être atteinte. Ces impératifs peuvent donc être appelés en général des impératifs de l'HABILETÉ. Que la fin soit raisonnable et bonne, ce n'est pas du tout de cela qu'il s'agit ici, mais seulement de ce qu'il faut faire pour l'atteindre. Les prescriptions que doit suivre le médecin pour guérir radicalement son homme, celles que doit suivre un empoisonneur pour le tuer à coup sûr, sont d'égale valeur, en tant qu'elles leur servent les unes et les autres à accomplir parfaitement leurs visées. […] On pourrait les désigner en disant : ce sont des *règles* (*Regeln*) de l'habileté […] On pourrait encore appeler [ces] impératifs *techniques* (se rapportant à l'art)[4].

On remarquera, au début de la citation, la présence de la thématique médiévale concernant la « partie pratique » des sciences[5], et plus loin celle de l'exemple aristotélicien du médecin et de l'empoisonneur[6]. La doctrine kantienne de la technique

4. *Fondements de la Métaphysique des Mœurs*, Ak. IV, 415 (p. 86 de la traduction française d'A. Philonenko). Nous nous autoriserons de ces propositions terminologiques kantiennes pour appeler ces règles *règles techniques* ou *règles de l'habileté*. Ces termes seront synonymes dans ce développement.
5. Voir plus haut, § 111.
6. *Mét.* Θ.2, 1046b7.

ne va pas beaucoup plus loin que cela. Kant y consacre peu de développements, signe que ce sujet devait lui sembler peu problématique. Cela est confirmé par le peu de débats que ces idées provoquèrent – alors que les sujets connexes de la raison théorique, de la loi morale, et même de la recherche du bonheur enflammèrent le post-kantisme. Kant et ses contemporains ne pensaient pas qu'il y avait rupture sur ce sujet avec la tradition remontant à la scolastique.

Dans la formulation kantienne cependant un progrès décisif est accompli, qui permet de percevoir pourquoi une procédure pratique est convertible en connaissance théorique. En la pensant comme *règle* en effet, Kant se donne la possibilité de considérer que *toute règle pour l'action se résout, en définitive, en une règle pour le jugement*.

Cette connexion est visible dans le petit opuscule *Sur le lieu commun : Il se peut que ce soit juste en théorie, mais en pratique ça ne vaut rien*[7] qui, s'il traite surtout de questions politiques, est introduit par quelques considérations générales sur la technique. Là, dans un passage célèbre, Kant se moque de l'artilleur qui hausserait les épaules face aux calculs de la balistique :

> Personne ne peut par conséquent se targuer d'être versé pratiquement dans une science et de mépriser pourtant la théorie sans montrer tout simplement qu'il est un ignorant dans sa discipline […] On ne ferait que rire d'un mécanicien empirique ou d'un artilleur qui voudrait critiquer la mécanique générale ou la doctrine mathématique des projectiles en arguant que la théorie en est certes subtilement conçue, mais qu'elle n'est toutefois d'aucune valeur dans la pratique puisque, dans l'application, l'expérience fournit des résultats tout différents de ceux de la théorie, (car si l'on ajoutait à la première la théorie du frottement et à la seconde celle de la résistance de l'air, si, par conséquent, on ne faisait que leur ajouter encore davantage de théorie, elles s'accorderaient parfaitement avec l'expérience)[8].

L'artilleur n'a donc pas à rejeter la théorie parce qu'elle ne s'accorde pas avec la pratique, mais au contraire il doit *juger*, en fonction des caractéristiques du tir qu'il prépare, quelles parties de la théorie prendre en compte, quelles sont les approximations acceptables ou pas, etc. S'il y a du vent, il ne pourra par exemple négliger de prendre en compte la résistance de l'air.

> Qu'entre la théorie et la praxis il faille encore un moyen terme pour établir la liaison et le passage de l'une à l'autre, et ce, quelle que soit la complétude de la théorie, c'est là ce qui va de soi ; *car au concept de l'entendement qui contient la règle, il*

7. (Kant [1785] 1987), Ak. VIII, p. 275.
8. *Sur le lieu commun...*, Ak. VI. 276.

> *faut que s'ajoute un acte de la faculté de juger par lequel le praticien décide si le cas tombe sous la règle ou non*[9]. *(Nous soulignons)*

La règle est ce qui fait le lien entre la théorie et la pratique, car l'acte propre du praticien est de juger quoi faire dans une situation donnée selon la règle fournie par l'entendement. Le texte continue en effet :

> Et comme on ne peut toujours à nouveau donner à la faculté de juger des règles selon lesquelles elle aurait à se guider dans la subsomption (car cela irait à l'infini) on conçoit qu'il puisse exister des théoriciens qui dans leur vie ne peuvent jamais passer à la pratique parce que leur manque la capacité de [bien] juger, par exemple des médecins ou des juristes qui ont bien fait leurs études mais qui ne savent pas comment ils doivent se comporter lorsqu'ils ont un conseil à donner[10].

On retrouve dans ce dernier exemple un écho à l'affirmation aristotélicienne de la supériorité de l'homme d'expérience sur le technicien « théorique » qui ne sait pas appliquer son savoir. C'est donc le jugement et l'expérience qui constituent le fond du savoir technico-pratique : savoir quel savoir mobiliser, en telle ou telle circonstance, savoir quand appliquer ou ne pas appliquer la règle. Mais le savoir positif qui est mobilisé, c'est-à-dire la règle elle-même, doit être considérée comme entièrement du ressort de l'entendement.

Or ce qui permet une telle annexion, c'est que les règles ne sont nullement pour Kant caractéristiques du seul savoir pratique : elles sont bien plutôt l'objet même de l'activité discursive en général :

> Si l'entendement est défini comme le pouvoir des règles, la faculté de juger est le pouvoir de *subsumer* sous des règles, c'est-à-dire de distinguer si quelque chose s'inscrit ou non sous une règle donnée[11].

Le caractère central et novateur de la notion de *règle* dans la théorie kantienne de la connaissance a été de nombreuses fois souligné, notamment dans le commentaire anglo-saxon[12]. Nous allons rappeler cette doctrine en suivant, quant à nous, l'interprétation de Béatrice Longuenesse (1993), qui a renouvelé en profondeur les études kantiennes.

9. *idem*, Ak. VI. 275.
10. *idem*.
11. *Critique de la raison pure*, A 132 / B 171, Ak. III, 131.
12. Voir par exemple (Stern 1974).

§ 132. Le concept comme règle du jugement

Longuenesse explicite très clairement que la caractérisation par Kant du concept comme règle doit être entendue en un double sens :

> En premier lieu, le concept est « règle » comme conscience de l'unité d'un acte de synthèse sensible, ou conscience du procédé d'engendrement d'une intuition sensible : ce premier sens de la règle anticipe ce que Kant, dans le chapitre consacré au Schématisme des concepts purs de l'entendement, appellera un schème ([19]A 140/B 180). Mais la règle a aussi un sens discursif : le concept est règle en ce que tout objet pensé sous ce concept se voit attribuer, comme caractères lui revenant nécessairement, les caractères définissant le concept sous lequel il est pensé[13].

Le premier point se réfère au jugement *déterminant*, comme l'appelle Kant[14], qui applique les règles fournies par le concept déjà formé à la situation présente :

> Si une matière est rouge et lourde, c'est du cinabre.
> Ceci a ces caractéristiques.
> *Donc :* ceci est du cinabre.

Le second point est illustré par notre capacité à déduire une nouvelle règle de règles déjà existantes :

> Si *x* est un animal, alors il est mortel.
> Si *x* est un homme, alors il est animal.
> *Donc :* Si *x* est un homme, alors il est mortel.

Nous avons exprimé de manière hypothétique les propositions de ce syllogisme, plutôt que de la manière catégorique traditionnelle (*Tous les animaux, etc.*), afin de bien montrer leur statut de règle. Kant est explicite à ce sujet :

> Quand je dis : « un corps est divisible », cela signifie : « quelque chose x, que je connais par les prédicats qui constituent ensemble le concept de corps, je le pense aussi par le prédicat de divisibilité »[15].

Ce point, sur lequel nous reviendrons tout à l'heure, permet de souligner que la liaison « simplement logique » des concepts entre eux doit toujours, chez Kant, être fondée en référence à l'objet *x* auquel elle doit pouvoir être appliquée dans l'expérience. La discursivité, qui nous permet d'organiser nos concepts en un tout cohérent par la dérivation de règles entre elles, doit toujours viser leur applicabilité à l'expérience, sous peine de s'égarer dans les considérations abstraites et vides de

13. (Longuenesse 1993, p. 46-47).
14. *Critique de la Faculté de Juger, Introduction*, § IV, Ak. V, 179.
15. Refl. 4634 (1772-1776), Ak. XVII, 616, citée par Longuenesse (ibid., p. 126).

la dialectique de la raison pure, comme l'explique Longuenesse :

> Derrière la figure visible ou audible du jugement, c'est-à-dire derrière sa figure grammaticale, se profile l'exercice véritable de la pensée dans lequel tout concept est rapporté, comme prédicat, au véritable sujet du jugement qu'est en dernière instance l'objet = *x*. […]
>
> D'un côté, l'« *x* » auquel il faut rapporter en dernière analyse la liaison des concepts dans le jugement peut être identifié à l'intuition sensible dans laquelle l'imagination a dessiné les schèmes *a priori* et *a posteriori* (associatifs) que réfléchissent concepts et jugements. Mais [l'intuition est elle-même l'objet du jugement] dans la mesure seulement où elle est rapportée à l'*x* pensé dans le jugement (l'objet auquel sont rapportés les concepts liés dans le jugement, et auquel est aussi, par là même, rapportée l'intuition sensible elle-même)[16].

En d'autres termes, les règles que sont les concepts théoriques non seulement permettent d'organiser, par leur maillage, le flux de l'expérience sensible, mais également elles sont immédiatement *pensées* comme objectives, c'est-à-dire comme n'étant pas dues à ma simple ingéniosité classificatrice, mais comme explicatives de la régularité même que j'observe dans l'expérience.

La formation des concepts et des règles est l'objet du jugement *réfléchissant* dans lequel Longuenesse reconnaît le processus « de réflexion, comparaison, abstraction », décrit dans la *Critique de la raison pure*[17]. Lorsqu'on rencontre des phénomènes qui résistent à notre compréhension immédiate, c'est-à-dire pour lesquels on ne trouve pas évidemment de règle à laquelle les subsumer, l'entendement prend une attitude réflexive, celle d'un « jugement silencieux » selon l'expression de Steckelmacher reprise par Longuenesse (1993, p. 146), visant à formuler la règle adéquate à cette expérience. Cette opération requiert que plusieurs phénomènes soient *comparés* entre eux afin qu'on en dégage, par la *réflexion*, la règle commune (par exemple le fait, pour le pin, le saule et le tilleul, d'avoir tous trois un tronc, qui devient donc un caractère du concept d'arbre sous lequel nous les rangeons) et que leurs différences en soient au contraire *abstraites*.

Cette théorie cognitive provient en droite ligne de la tradition associationniste de Locke, Hume et Berkeley. Cependant elle s'en distingue par deux traits essentiels[18] : d'une part, contre Hume et Berkeley, le concept d'arbre n'est pas une

16. (Longuenesse 1993, p. 127, 130).
17. *Critique de la faculté de juger*, Introduction, IV, Ak. V, 179. Sur les termes *déterminant* / *réfléchissant*, voir (ibid., p. 135, n. 1).
18. (ibid., p. 141-143).

image, une idée particulière à laquelle nous conférons une valeur générale ; il est bien une idée générale, comme le voulait Locke ; et d'autre part contre Hume et Locke, on ne doit pas dire que le général n'est pas dans les choses, mais uniquement dans la manière dont nous les pensons :

> L'universel appartient bel et bien à l'existence des choses, à condition d'y être dessiné comme règle [...] Tout factice qu'il soit quant à sa forme, un concept représente un universel « en soi présent » dans l'objet donné parce que « en soi présent » dans notre appréhension de l'objet donné[19].

Autrement dit, la simple association empirique des perceptions que réalise l'imagination reproductive, et qui nous est sans doute commune avec les animaux, est élevée par le concept au rang de *règle*, c'est-à-dire qu'elle est pensée comme propriété de la réalité elle-même, et donc *comme nécessaire*[20].

Ainsi, dans la doctrine kantienne, le concept, comme représentation générale, se trouve encadré par deux actes : le jugement réfléchissant qui *produit* le concept, et le jugement déterminant qui subsume le divers de l'expérience sous le concept, donc qui *utilise* le concept déjà formé. Le concept est le résultat d'une synthèse et rend possible des synthèses subséquentes. Cette structure permet de clarifier l'ambiguïté qui pèse souvent sur les assertions kantiennes concernant le jugement. Ce terme désigne d'abord deux *actes* de nature différente, l'acte de former un concept et l'acte de l'appliquer à l'expérience ou à un raisonnement. Mais ce n'est pas tout, il désigne également les représentations qui résultent de ces actes et qui s'expriment par des propositions. Kant appelle jugement aussi bien l'acte de former la proposition « Ceci est du cinabre », que cette proposition elle-même. Cette seconde ambiguïté entache également les autres termes utilisés par Kant pour désigner le jugement : on entend plutôt l'acte dans le terme *synthèse*, et plutôt le résultat de l'acte dans le terme *liaison*, quoique l'autre interprétation soit également possible dans chaque cas. Or cette ambiguïté est essentielle, car elle dénote le fait que juger est un acte qui mène vers un état accompli (comme « écrire une lettre ») plutôt qu'une simple activité sans résultat (comme « courir »). Le jugement, dans son expression propositionnelle, énonce la règle qui a été élaborée par le jugement réfléchissant et inscrite dans le concept, et qui peut être à nouveau extraite du concept pour former la majeure d'un jugement déterminant[21].

Afin de souligner l'actualité de cette position, il nous semble utile de la rap-

19. (ibid., p. 143).
20. (ibid., p. 40, 46).
21. *Prolégomènes à toute Métaphysique future*, §23, Ak. IV, 305.

procher des doctrines inférentialistes de la signification, qui se sont développées depuis les années 1980, et dont Osta Vélez (2020) dresse un panorama. Ces thèses peuvent être opposées à une conception qu'Osta Vélez appelle « logiciste », et qui tient que les concepts sont des composants « inertes » des raisonnements, eux-mêmes conduits selon les seules lois de la logique. Le seul rôle des concepts serait de dénoter des objets de l'expérience, et il y aurait ainsi une division du travail nette entre la syntaxe qui fournirait les règles de formation et de transformation du langage, et la sémantique qui s'occuperait seulement de la dénotation. Les thèses inférentialistes récusent une telle division et affirment au contraire que la signification des concepts est donnée, au moins partiellement, sinon totalement, par leurs règles d'usage, et notamment par les inférences qu'ils autorisent. Il faut entendre par *inférence* l'exercice de la raison en général, qui établit une représentation *A* à partir d'une situation *S* (qui peut elle-même être propositionnelle) – bien au-delà de la seule déduction, qui a été le mode d'inférence privilégié dans la réflexion philosophique, et qui a conduit à la position logiciste.

§ 133. La structure formelle des règles du jugement

Il est à présent possible d'analyser pourquoi la structure des règles peut être élucidée à partir de l'analyse des formes logiques du jugement, *i.e.* des propositions qui expriment les contenus des concepts : une règle, pour Kant, énonce une relation entre une condition et sa conséquence. Cette relation, selon la table logique des jugements qu'il propose[22], est de trois sortes : catégorique, hypothétique ou disjonctive.

On peut cependant s'étonner de cette tripartition à deux titres.

D'abord, elle ne suit pas l'usage de la logique classique. Par exemple la *Logique* de Port-Royal distingue les jugements simples (que Kant appelle catégoriques) des jugements composés, et parmi ceux-ci, outre les jugements hypothétiques et disjonctifs, reconnaît de nombreuses autres formes, comme les conjonctifs, les concessifs, etc. À cela Longuenesse répond de manière convaincante que ce choix, parmi les nombreux autres qui lui étaient proposés, démontre que Kant n'est nullement tributaire de la forme imparfaite de la logique de son temps, comme Cavaillès le lui a entre autres reproché, mais qu'au contraire il y met en exergue ce qui, selon lui, dérive essentiellement de la structure transcendantale du sujet. Comme elle

22. *Critique de la raison pure*, A 70 / B 95, Ak. III, 87.

l'écrit :

> L'exposé des formes de la pensée n'est tributaire ni d'une introspection empirique, ni d'une déduction à partir de l'identité vide, du simple « accord de la pensée avec elle-même ». Il est au contraire ordonné à ce que la critique a révélé être la fonction essentielle de la pensée discursive : établir « au moyen de l'unité analytique » l'unité originairement synthétique de l'aperception[23].

Ensuite, si le jugement est une règle, c'est la forme hypothétique qui semble lui convenir naturellement. On ne comprend pas aisément en quoi le jugement catégorique pourrait être une règle, puisqu'il semble affirmer inconditionnellement un fait. Quant au jugement disjonctif, il semble n'être qu'une conjonction de deux jugements hypothétiques réciproques (*Si ce n'est A, c'est B, et si ce n'est B, c'est A*).

Concernant le jugement catégorique, on a vu plus haut qu'il pouvait être exprimé comme un jugement hypothétique en mettant en valeur l'objet en question (*Les corps sont divisibles = Si un objet x est un corps, alors x est divisible*). Kant tient cette convertibilité, comme le rappelle Longuenesse, de la logique wolffienne[24]. Mais dans ce cas, on peut se demander à plus forte raison pourquoi donc Kant les distingue l'un de l'autre. La réponse se trouve, selon Longuenesse, dans l'*Amphibologie des concepts de la réflexion*, qui clôt l'*Analytique transcendantale*[25]. Dans ce texte, le couple de l'intérieur et de l'extérieur est associé aux catégories de la relation. Longuenesse met en évidence leur rôle logique dans la formation des jugements ; ils sont les concepts essentiels par lesquels nous comparons les phénomènes entre eux afin de délimiter le périmètre de l'objet. Ainsi, une condition qui est jugée intérieure à l'objet donne lieu à un jugement catégorique, ainsi : *« Si x est un arbre, alors x a un tronc »*, car le concept de *tronc* est partie de la synthèse par laquelle je pense celui d'*arbre*. Au contraire, l'idée de *congélation* de l'eau suppose une condition extérieure à l'eau, celle d'une basse température. Dans *« Si la température est basse, l'eau gèle »*, le jugement est naturellement exprimé sous forme hypothétique.

Le fait que dans nos jugements le curseur entre l'intérieur et l'extérieur soit mobile répond à la convertibilité de tout jugement hypothétique en jugement catégorique. Si le jugement considère non seulement l'eau, mais également le congélateur dans lequel on l'a placée, alors il peut affirmer catégoriquement qu'elle y

23. (Longuenesse 1993, p. 79-80).
24. (ibid., p. 109).
25. (ibid., pp. 170-185).

est gelée. Ce qui ultimement fixe le périmètre de nos concepts est la stabilité ou l'instabilité de ces rapports réglés à travers le temps, que nous auscultons *à partir* de nos couples de concepts *a priori* substance/attribut et cause/conséquence, qui ne sont rien d'autre que des règles pour former des règles, ou encore la forme générale des règles. Longuenesse cite à l'appui ces textes éclairants de Kant :

> Les déterminations intérieures d'une *substantia phaenomenon* dans l'espace ne sont que des rapports et elle-même n'est, en tout et pour tout, qu'un ensemble de pures relations. Nous ne connaissons la substance dans l'espace que par des forces qui sont actives en lui, soit pour y attirer d'autres substances (attraction), soit pour les empêcher d'y pénétrer (répulsion et impénétrabilité) (A 265/B 321).
>
> La matière est *substantia phaenomenon*. Ce qui lui convient intérieurement, je le cherche dans toutes les parties de l'espace qu'elle occupe, et dans tous les effets qu'elle produit, et qui assurément ne peuvent jamais être que des phénomènes des sens extérieurs. Je n'ai donc rien qui soit absolument intérieur, mais quelque chose qui ne l'est que comparativement, et qui lui-même se compose de rapports extérieurs (A 277/B 333)[26].

Cette analyse permet de comprendre comment appliquer la différence entre jugement catégorique et hypothétique aux règles techniques. Un technicien peut énoncer des jugements catégoriques quant aux conséquences qui dérivent directement *de sa propre action* – s'il ne fait qu'énoncer ce qu'il entendait faire. Par exemple, lorsqu'il affirme la verticalité d'un mur après l'avoir mesurée avec un fil à plomb, il ne fait que développer ce que nous entendons communément par verticalité, qui se constate par la trajectoire de chute d'un objet. Bien entendu, si nous nous trouvions sur une planète ayant une vitesse de rotation importante, une telle définition devrait être revue, mais dans un jugement catégorique on considère justement le contexte externe à l'objet *fixe*. Lorsqu'il prévoit par contre que le mur résistera à des vents jusqu'à 190 km/h, il énonce un jugement hypothétique, car il explicite une condition externe à la résistance du mur. Cette assertion peut elle-même être justifiée par d'autres jugements techniques sous-jacents, qui portent par exemple sur les matériaux employés et les techniques de construction. Ces jugements peuvent également être catégoriques lorsqu'ils portent sur l'action du technicien (« *ce mur résistera parce que je l'ai construit avec des briques et un ciment qui ont cette propriété* ») ou hypothétiques lorsqu'ils portent sur des conditions externes (« *si les briques qu'on m'a donné sont bien celles que j'avais spécifiées* »). En d'autres termes, tandis que les jugements catégoriques techniques

26. (Longuenesse 1993, p. 174).

permettent d'expliciter les conséquences attendues d'une action, les jugements hypothétiques explicitent ses conditions externes de succès.

Le jugement disjonctif a un rôle différent des deux premiers. Kant le décrit ainsi :

> Le jugement disjonctif contient un rapport de deux ou plusieurs propositions entre elles ; non pas un rapport de conséquence, mais un rapport d'opposition logique, dans la mesure où la sphère de l'une exclut la sphère de l'autre ; mais pourtant, en même temps, un rapport de communauté, en ce qu'elles remplissent ensemble la sphère de la connaissance considérée ; et donc un rapport des parties de la sphère d'une connaissance où la sphère de chacune des parties est un complément de la sphère de l'autre pour constituer le concept complet de la connaissance divisée[27].

Par *sphère* de la connaissance il faut entendre ce que nous appellerions aujourd'hui l'extension de la proposition, les *cas* dans l'expérience où elle se trouve être valide[28]. Le jugement disjonctif vise à déterminer l'ensemble des conditions pertinentes à la description réglée d'un phénomène, c'est-à-dire la *systématicité*. Aussi faut-il, pour décrire les états possibles de l'eau, distinguer ses températures critiques de congélation et de vaporisation ; ou encore, pour décrire toutes les trajectoires possibles d'un corps massif, le considérer comme partie d'un système plus large incluant tous les autres corps exerçant une attraction sur lui. Dans ce second cas, il faut reconnaître qu'il y a *co-détermination* des conditions entre elles, puisque le corps étudié exerce également des forces sur les autres parties du système. Il y a ici, comme souvent lorsqu'on recherche la systématicité, une re-conceptualisation du problème : les conditions ne sont plus représentées par les corps agissant sur le corps étudié, mais par les conditions initiales du système dans son ensemble, qui déterminent toutes ses trajectoires possibles. La recherche de règles disjonctives dans la formation d'un concept permet donc, en quelque sorte, de calibrer son périmètre même et de reformuler lesquelles de ses règles valent catégoriquement et lesquelles hypothétiquement, c'est-à-dire sous condition extérieure.

Il convient ici de relever l'originalité de la conception kantienne. L'association de la systématicité à la division exhaustive d'un concept remonte à Platon[29], et se popularise aux Temps modernes par le diagramme de l'arborescence[30] ; l'idée que ce concept créée ainsi un *espace* qu'il est possible de parcourir complètement

27. *Critique de la raison pure*, A 74/ B 99, Ak. III, 89.
28. (Longuenesse 1993, p. 437).
29. Voir plus haut, § 107.
30. Voir plus haut, § 114.

et sans répétition est aussi commune. L'originalité de Kant consiste à considérer que la division ne concerne pas d'abord les prédicats du concept, mais son extension, c'est-à-dire directement les phénomènes de l'expérience que le concept doit subsumer[31]. Le jugement disjonctif donne la règle qui permet d'explorer l'environnement du concept et d'identifier *tous les cas* où la règle s'appliquera ou non, afin de réorganiser le concept en conséquence.

Les jugements disjonctifs sont donc à l'œuvre de manière prioritaire dans la démarche scientifique. Lorsque les conditions d'une règle ne peuvent, *après analyse*, être pensées autrement que comme intérieures au concept visé, cette règle doit être alors qualifiée d'analytique, c'est-à-dire comme constitutive logiquement du concept. Ainsi, le jugement que tout arbre a un tronc, même s'il fut formé initialement de manière synthétique et empirique, peut être considéré comme analytique lorsque la réflexion scientifique a jugé, après examen d'une grande variété d'arbres, et surtout après modélisation de leurs fonctions internes et de leur structure biologique, que cet élément est partie intégrante de ce modèle. Le tronc n'est rien d'autre que la tige de l'arbre, nous apprend la botanique, et s'il y a des plantes acaules (sans tige), leur ordre est exclusif de ceux sous lesquels sont rangées les différentes espèces d'arbres[32]. Que l'arbre ait un tronc résulte donc des conventions par lesquelles nous avons décidé de décrire les espèces végétales. Par contraste, il existe des arbres sans feuilles (les résineux) et le jugement « *L'arbre a des feuilles* » n'est donc pas analytique. Cela ne veut pas dire que ces conventions sont arbitraires et subjectives. Ce qui est objectif, ce sont les règles que l'entendement a discerné à travers la masse des phénomènes. Leur regroupement en « paquets » que sont les modèles et les concepts est l'affaire de la mise en ordre systématique que visent les règles disjonctives, et c'est celles-ci qui, ultimement, déterminent l'analyticité. L'analyticité n'est donc pas un caractère propre aux jugements, mais il est relatif aux systèmes conceptuels dans lesquels ils sont plongés. *A contrario*, la plupart des propositions catégoriques du langage courant sont bien synthétiques et, après analyse, peuvent toujours être réexprimées sous forme hypothétique. Elles ne sont vraies que sous certaines conditions qui ne sont pas explicitées, soit parce qu'elles valent toujours ou la plupart du temps pour le locuteur qui les énonce, soit parce qu'elles ont été tout simplement oubliées. Ainsi, par exemple, la proposition : « Il n'y a plus d'animaux dangereux dans les forêts » est vraie si on lui ajoute la condi-

31. (Longuenesse 1993, p. 439-441).
32. Nous nous basons ici, très modestement, sur Wikipédia, articles *tronc, tige, plante acaule*.

tion : « Si la forêt est en Europe de l'Ouest ».

Mais les jugements disjonctifs ont également le même rôle dans la démarche technique qui est, on se rappelle, d'universaliser une délibération, c'est-à-dire d'expliciter les conditions de validité d'une action achevant une fin souhaitée[33]. Ces conditions sont les paramètres *externes* de succès d'une action, qui permettent également d'envisager toutes ses variantes possibles. En d'autres termes, il s'agit de construire *l'espace de problème* de l'action technique, au sein de laquelle elle peut se déployer de manière assurée – ou encore, dans le modèle contractuel de la programmation, d'expliciter les conditions d'environnement *E* requises pour que la spécification soit effective[34].

L'interprétation de Longuenesse nous permet donc d'avancer que le choix par Kant de la trilogie catégorique / hypothétique / disjonctif fut guidé par une réflexion profonde sur la notion de règle et sur la relation de condition à conséquence. Cette trilogie est animée par la dynamique interne que lui impose le couple interne / externe. Ce couple lui-même dérive immédiatement de l'acte même du jugement réfléchissant, qui discrimine à travers la masse des phénomènes ceux qu'il associe au noyau du concept en cours d'élaboration et ceux qu'il considère comme étant en simple relation réglée avec lui.

Cette interprétation s'applique de manière tout à fait naturelle aux jugements techniques, selon qu'ils portent sur le concept même de l'action – qui contient toujours la visée de ses conséquences – ou sur les conditions matérielles de son succès. Le jugement disjonctif, quant à lui, joue le même rôle dans la démarche technique qu'en sciences, d'identifier de manière systématique ces conditions externes afin de dessiner un espace dans lequel l'action peut être entreprise avec certitude.

§ 134. Le raisonnement comme jugement complexe

À la convertibilité qu'elle permet de penser entre jugement catégorique et jugement hypothétique, c'est-à-dire comme nous l'avons vu, entre intérieur et extérieur d'un concept, on doit ajouter une seconde forme de conversion, qui traverse la logique tout entière, entre *concept, jugement* et *syllogisme*, et où « la forme discursive centrale est […] le *jugement*, dans la définition duquel se trouvent inscrits aussi bien les *concepts* que les *raisonnements*[35] ».

33. Voir plus haut, § 99.
34. Voir plus haut, § 54.
35. (Longuenesse 1993, p. 80).

Concernant la convertibilité entre concept et jugement, elle se justifie par le fait que le jugement est l'acte qui forme, en amont, la représentation qu'est le concept, ou qui l'applique en aval à l'expérience ou à un raisonnement. Le concept, formé au confluent de plusieurs actes réfléchissants, est comme un « paquetage » de règles : on peut voir dans l'arbre le tronc, mais également les fruits, les racines, et quantité d'autres propriétés. Chacune d'elles peut être exprimée par une règle, que le concept assemble en une unité. Les jugements déterminants, quant à eux, « déplient » ce paquetage (les philosophes anglo-saxons utiliseraient ici volontiers l'expression *unpack the meaning of...*) en des jugements singuliers. Le terme « paquetage » n'est cependant pas entièrement approprié, car il suggère que cet assemblage n'est pas structuré. C'est justement le travail de la réflexion scientifique que d'établir une liaison systémique entre ces règles. Par exemple, identifier la formule chimique du cinabre (le mercure) permet d'expliquer pourquoi il est à la fois rouge et pesant, et de transformer ce paquetage empirique en un système cohérent, qui permet d'en dériver toutes les autres propriétés. Le concept, une fois le travail scientifique achevé, est un *système de règles*.

Concernant la convertibilité entre jugement et syllogisme, si la règle est la majeure du syllogisme, il est tout aussi vrai de remarquer que le syllogisme n'est rien d'autre que l'application de la règle. Comme le dit Longuenesse :

> Le syllogisme, considéré comme inscription d'un jugement donné sous une règle générale qui en donne la raison, ou usage d'un jugement comme règle pour un autre jugement, n'est pas une fonction distincte de celle du jugement et venant s'y surajouter. Il est au contraire une fonction en quelque sorte « enveloppée » dans tout jugement[36].

Le raisonnement n'est rien d'autre que le pouvoir de juger médiatement, c'est-à-dire d'enchaîner l'application des règles de manière à ce que leurs conditions s'y trouvent résolues :

> La raison considérée comme le pouvoir de donner une certaine forme logique à la connaissance est le pouvoir d'inférer, c'est-à-dire de juger médiatement (en subsumant la condition d'un jugement possible sous la condition d'un jugement donné)[37].

36. (Longuenesse 1993, p. 109).

37. « Vernunft, als Vermögen einer gewissen logischen Form der Erkenntniß betrachtet, ist das Vermögen zu schließen, d. i. mittelbar (durch die Subsumtion der Bedingung eines möglichen Urtheils unter die Bedingung eines gegebenen) zu urtheilen. » Nous corrigeons ici l'édition de la *Pléiade* qui traduit *mittelbar* par *immédiatement*, ce qui est bien sûr le contraire de ce que veut dire Kant.

> Le jugement donc est la règle générale (la majeure, *major*). La subsomption de la condition d'un autre jugement possible sous la condition de la règle est la mineure (*minor*). Enfin, le jugement réel qui exprime l'assertion de la règle dans le cas subsumé est la conclusion (*conclusio*).[…] On voit aisément que la raison arrive à une connaissance par des actes de l'entendement qui constituent une série de conditions. […] Or, toute série dont l'exposant (que ce soit d'un jugement catégorique ou hypothétique) est donné pouvant être poursuivie, ce même acte de la raison conduit par suite à la *ratiocinatio polysyllogistica*, laquelle est une série de raisonnements qui peut être continuée sur une étendue indéterminée, soit du côté des conditions (*per prosyllogismos*), soit du côté du conditionné (*per episyllogismos*)[38].

Nous avons cité ce texte en longueur car il montre avec force que Kant ne conçoit pas le syllogisme comme la composition de jugements, et le raisonnement comme la composition de syllogismes, où le niveau inférieur de synthèse constituerait la « matière » pour le niveau supérieur, selon le schéma promu entre autres par la *Logique de Port-Royal*. Au contraire, c'est le syllogisme et le raisonnement qui sont convoqués par le jugement, c'est-à-dire par la visée de l'entendement qui cherche à appliquer la règle au cas présent. L'unité du jugement qui cherche à se déterminer précède les séries et les réseaux de jugements médiats. Elle les organise – ou encore les *ordonnance*, peut-on dire en référence à l'*ordinatio* médiévale – en vue de son propre achèvement.

Cette idée d'ordonnancement permet de percevoir la proximité de la notion kantienne de jugement complexe de celle moderne de *procédure* technique. Il est utile pour cela de considérer un jugement complexe qui vise à prendre une décision. Ainsi, une *check-list* de décollage ne dit pas à l'aviateur *comment* décoller, mais *s'il peut* le faire. Elle ressemble à ceci :

> Pompe à carburant auxiliaire — Désactivée
> Commandes de vol — Libres et dans le bon sens
> Instruments et radios — Vérifiés et réglés
> Feux de position du train d'atterrissage — Vérifiés
> Altimètre — Réglé
> Gyroscope directionnel — Réglé
> Jauges de carburant — Vérifiées
> Ceintures de sécurité/harnais — Attachés
> Frein de stationnement — Désactivé

38. *Critique de la raison pure*, A 331 / B 388, Ak. III, 256.

[…][39]

On peut noter que cette *check-list* fait référence à d'autres *check-lists*. Par exemple, la vérification de la jauge de carburant comporte elle-même deux éléments principaux à vérifier : 1) que la jauge fonctionne correctement, qu'elle affiche la quantité réelle de carburant présente dans les réservoirs, 2) qu'il y a assez de carburant pour le vol qu'on entreprend. La vérification du premier point concernant le bon fonctionnement de la jauge est lui-même délicat et plusieurs procédures visent à s'en assurer. Il s'agit donc bien d'un jugement complexe au sens de Kant, en ce qu'il est constitué d'une conjonction de jugements partiels, qui eux-mêmes peuvent être complexes – de la même manière qu'une procédure peut faire appel à des sous-procédures. On s'approche ici de ce que nous appelons un *programme*, dont les actions sont des simples *jugements* – cependant il ne s'agit là encore que d'une remarque suggestive, que nous devrons expliciter dans la suite de notre travail.

8.2 L'ordre naturel et l'ordre intentionnel

Dans cette section, nous commençons (§ 135) par mettre en question l'identification, que Kant semble suggérer, entre règles du jugement et règles de l'action : elles ne sont évidentes ni pour la grammaire, ni pour le sens commun, ni pour la philosophie, ni pour la logique. Nous proposons une première explicitation de cette thèse à partir de l'analyse des impératifs hypothétiques (« *les règles de l'habileté* »[40]). Dans la suite de cette section et la section suivante, nous mettons en débat Kant et Aristote sur cette question, en reprenant les trois distinctions entre théorie et pratique par lesquelles nous avions conclu notre réflexion sur Aristote. Dans le développement § 136 nous étudions la première d'entre elles, celle qu'Aristote met le plus en avant, entre raison des objets contingents et raison des principes nécessaires. Nous nous appuyons pour ce faire sur la réflexion d'Elisabeth Anscombe qui objecte à Kant et aux modernes en général d'avoir mal compris le sens de cette distinction. Au développement suivant § 137, nous apportons une réponse kantienne à cette objection.

La section suivante est consacrée à l'étude des deux autres distinctions faites par Aristote, selon la modalité et la finalité de la connaissance.

39. www.aopa.org.
40. Voir plus haut, § 131.

§ 135. Règles, jugements et impératifs

La première section de ce chapitre a rappelé le rôle central de la notion de *règle* dans la doctrine kantienne de la connaissance et soutenu qu'elle était au fondement du statut dérivé du savoir technico-pratique à l'égard du savoir théorique. Cette affirmation se fondait sur quelques textes qui laissaient entendre qu'il s'agissait bien des *mêmes* règles qui, selon Kant, étaient capables de réguler le jugement aussi bien que la conduite humaine. Cependant en quel sens comprendre une telle thèse ? Car, pour le sens commun, les règles du jugement semblent bien être différentes des règles de l'action. La position kantienne signifie-t-elle qu'une telle distinction est seulement apparente ? Dewey affirme, par exemple, que la distinction entre sciences et techniques est propre à la culture occidentale, et qu'elle est issue des préjugés des aristocrates grecs contre le travail manuel[41]. Mais dans ce cas, comment faire droit aux différences évidentes que l'on perçoit entre les règles qui auraient pour objet l'action concrète et empirique d'un sujet agent, et celles qui porteraient sur l'activité intellectuelle d'un sujet pensant ? Qu'y a-t-il de commun entre juger que 3 et 5 font 8 et qu'il faut acheter un cheval ? Ou encore, entre juger que cette plante est un arbre, et qu'on peut traverser la rue ?

Kant se situe dans la droite ligne de la tradition classique issue de Jean Philopon[42], qui jugeait que le jugement du bon ou de l'utile venait se surajouter à celui de sa vérité, sans en modifier le contenu. Un impératif de la volonté tire son caractère prescriptif de ce qu'il est un jugement portant sur le caractère bon ou mauvais de l'état de choses représenté. Les impératifs, dit Kant, ne sont rien d'autre que des formules qui représentent « un principe objectif, en tant qu'il est contraignant pour une volonté. […] Ils disent qu'il serait bon de faire telle chose ou de s'en abstenir ; mais ils le disent à une volonté qui ne fait pas toujours » ce qui lui présenté comme bon[43].

Les distinctions qui s'appliquent aux jugements valent donc également pour les impératifs. Ainsi, un impératif est catégorique si la règle qui permet de conclure à sa bonté est tirée du concept même de l'objet jugé, et au contraire hypothétique si cette bonté dépend d'une condition externe. Cela veut dire que les impératifs hypothétiques « importent » la bonté de l'extérieur (*Si ceci est bon, alors cela l'est aussi*).

41. (Dewey [1925] 2012, p. 114).
42. Voir plus haut, § 111.
43. *Fondements de la Métaphysique des Moeurs*, Ak. IV, 413.

Or il n'y a nul impératif réellement catégorique hormis la loi morale – car il n'y a nulle chose intrinsèquement bonne hormis la volonté – Kant a suffisamment insisté là-dessus au début des *Fondements de la métaphysique des mœurs*. Par exemple, même la proposition « *Tout négociant souhaite s'enrichir* », qui semble seulement expliciter ce qu'est un négociant, peut être l'objet d'une condition. Il se pourrait en effet que le négociant travaille pour une ONG qui propose un programme de soutien à l'agriculture locale : dans ce cas il refusera des pratiques spéculatives, et s'engagera plutôt sur des décisions pluri-annuelles garantissant les agriculteurs contre les aléas du climat. D'une manière générale, Kant tient qu'il est de l'essence d'une volonté libre de ne devoir jamais être asservie à aucun commandement catégorique, si ce n'est celui qui lui enjoint, justement, d'être autonome. Il n'est aucun rôle, aucun métier, aucune communauté d'appartenance auxquels notre personne pourrait être si attachée que nous devrions nous soumettre *en tous cas* à leur logique interne.

La convertibilité des impératifs techniques en propositions théoriques concerne donc seulement les règles hypothétiques. Ils ne sont justifiés, comme tous les jugements de ce type, par rien d'autre que la liaison entre condition et conséquence. Ainsi, la proposition « *Tous les négociants en denrées agricoles basent leurs décisions sur les prévisions météorologiques* » a une apparence catégorique qui peut être interprétée aussi bien comme un constat que comme un impératif (« *Négocie les denrées agricoles en fonction des prévisions météo !* »). Mais ces formes catégoriques, qu'elles soient déclaratives ou impératives, ne sont que des manières apparentes d'exprimer un unique jugement : *Si on veut réaliser des négociations profitables en denrées agricoles, alors il faut consulter les prévisions météo*. Ce jugement contient d'ailleurs de nombreuses autres conditions implicites qu'il revient à l'analyse théorique d'expliciter, comme par exemple : « *si le négociant veut s'enrichir* », ou « *si les denrées en question ne sont pas cultivées sous serre* », etc.

Ainsi les critères de validité d'un jugement et d'un impératif hypothétiques sont identiques : ils résident tous deux dans la connexion nécessaire entre condition et conséquence. D'ailleurs, le mouvement *théorique* d'explicitation des conditions externes d'un jugement correspond exactement au mouvement de la *spécification* technique qui étend le domaine de validité d'une délibération en précisant les moyens nécessaires et les conditions d'efficacité de la procédure. Qu'il s'agisse donc d'un simple constat ou d'une préconisation d'action, étendre l'universalité d'un jugement semble bien relever d'une même faculté d'élaboration de règles, la

faculté de juger *réfléchissante* de l'entendement[44].

Parce qu'impératifs et jugements se fondent *sur les mêmes règles*, les trois caractères par lesquels Aristote pensait pouvoir distinguer génériquement la connaissance pratique de la connaissance théorique apparaissent finalement tout relatifs : que l'objet de la règle soit naturel ou artificiel, que celle-ci soit mobilisée pour former un jugement ou une décision, et enfin que la visée du jugement ou de la décision soit le savoir désintéressé ou non – tout cela ne change rien à la règle elle-même, et donc rien non plus à la connaissance. C'est cet argument général que nous allons suivre dans les développements suivants, en rendant la parole à Aristote, par la voix de quelques philosophes contemporains.

§ 136. L'ordre intentionnel

Pour commencer par la première distinction, Thomas d'Aquin et Kant objectent à Aristote qu'il est tout à fait possible de considérer les objets contingents – les artefacts – spéculativement, c'est-à-dire comme des objets naturels. Ainsi, on peut étudier une maison du point de vue des lois de la mécanique tout comme on le ferait pour n'importe quelle structure naturelle.

À cela Aristote répond qu'en étudiant ainsi la maison, on ne l'étudie pas *en tant qu'objet de la délibération* (par exemple savoir si elle est habitable), mais en tant qu'une certaine structure ayant telle résistance au vent, ou en tant qu'une certaine masse de pierres, etc. Ce que ne peut faire l'esprit théorique, c'est raisonner à propos des intentions elles-mêmes. Or ce type de raisonnement, nous l'avons vu lors de notre étude d'Aristote, a des propriétés spécifiques, car il ne porte pas seulement sur la recherche et l'évaluation des solutions, mais également sur la fixation de leurs critères, c'est-à-dire sur la spécification même de *ce qu'il faut vouloir*, en précisant les ressources disponibles (argent, matériaux, temps de construction, etc.) et les contraintes (tant de personnes à loger, garage, jardin, efficacité énergétique, etc.).

La réponse rationaliste à cette objection, celle que fait Simon[45], est que si la spécification du problème doit être elle-même le résultat d'une réflexion *rationnelle* (et non par exemple l'expression d'une préférence ou d'une intuition), alors elle doit être thématisée à son tour comme un problème. Arbitrer entre plusieurs

44. Sur la notion de jugement réfléchissant, voir plus haut § 132.
45. Voir plus haut, § 65.

contraintes de l'espace de problème, c'est les rapporter à des utilités plus lointaines et donc les plonger dans un « méta-espace de problème » plus large, etc.

Elizabeth Anscombe, dans son ouvrage célèbre tout empreint d'Aristote, *Intention*, prend le contre-pied exact de ce type d'argument, et affirme l'irréductibilité du syllogisme pratique au syllogisme théorique :

> Le rôle du « vouloir » dans le syllogisme pratique est très différent de celui d'une prémisse. Il est que ce qui est décrit dans la proposition constituant le point de départ de l'argument doit être voulu afin que le raisonnement mène à une quelconque action. Donc la forme « Je veux une vache de Jersey, ils en ont de bonnes au marché d'Hereford, donc je vais y aller » était formellement incorrecte : le raisonnement pratique devrait être donné sous la forme « Ils ont des vaches de Jersey au marché d'Hereford, donc je vais y aller[46].

En « décapitant » ainsi le syllogisme pratique de sa majeure, Anscombe veut montrer que la volonté n'est pas une hypothèse du raisonnement, mais sa raison d'être. On n'aurait jamais pensé à remarqué qu'il y avait des vaches de Jersey à Hereford si notre *intérêt* n'y avait prêté attention de prime abord. Elle se réfère ainsi explicitement à la notion d'ordre, dont elle a bien perçu la centralité dans les conceptions antiques et médiévales auxquelles elle veut revenir :

> Si la conception d'Aristote [du raisonnement pratique] consistait à décrire des processus mentaux effectifs, elle serait assez absurde. L'intérêt de cette conception est qu'elle décrit un *ordre* qui est là dès que des actions sont réalisées avec des intentions[47]. *(Nous soulignons)*

Anscombe utilise le terme *intention* pour désigner ce qui est en-deçà des processus psychologiques du désir et de la volonté. L'intention ne se limite pas en effet à ces formes supérieures de la « tentative d'obtenir » quelque chose (*trying to get...*) ; elle est commune à tous les animaux. L'intention n'est pas un phénomène, mais une manière de regarder les phénomènes. Elle est ce qu'on vise quand on pose la question « Pourquoi a-t-il fait cela ? », ou ce que recherchent les physiologistes quand ils étudient le comportement de l'animal ; ils distinguent en effet naturellement les mouvements du chien qui réagit à un bruit soudain, de ceux du chien qui cherche de la nourriture. Autrement dit, pour Anscombe, dans ces exemples l'intention n'est pas un objet de la description ou du raisonnement, mais leur raison d'être. Pour utiliser le vocabulaire de Kant contre lui-même, on pourrait dire qu'elle est comme une condition de possibilité de la raison pratique, et qu'elle

46. (Anscombe [1957] 2000, p. 66).
47. (ibid., p. 80).

appartient donc bien au domaine *transcendantal*. L'erreur de Kant aurait été de concevoir la volonté comme un objet naturel[48]. C'est ce qui lui aurait fait manquer de voir l'autonomie du domaine pratique (au sens usuel, non-kantien), et qui l'aurait conduit à le rapatrier dans le domaine théorique. La critique d'Anscombe vis-à-vis des Modernes s'applique en particulier à Kant :

> Se pourrait-il que la philosophie moderne ait radicalement manqué une chose : à savoir ce que les philosophes anciens et médiévaux entendaient par connaissance pratique ? Il est certain que dans la philosophie moderne, nous avons une conception incorrigiblement contemplative de la connaissance. Celle-ci doit être quelque chose qui est jugé comme tel par sa conformité aux faits. Les faits, la réalité, sont antérieurs et dictent ce qui doit être dit, s'il s'agit de connaissance. Et là se trouve l'explication de l'obscurité totale dans laquelle nous nous trouvons[49].

Afin de rendre plus claire cette idée que l'intention régit un ordre de la connaissance (« Le terme intentionnel fait référence à une *forme* de description des événements[50] ») irréductible à celui de la connaissance par observation, nous proposons de prendre un exemple plus parlant que celui des vaches de Hereford donné par Anscombe. Il est inspiré d'une anecdote relatée par Pline[51]. Celui-ci raconte que Démocrite, afin de prouver à ses concitoyens la supériorité de la science sur les occupations pratiques, avait spéculé avec succès sur le cours de l'huile d'olive grâce à sa connaissance des phénomènes météorologiques. Ayant montré que la science permettait de s'enrichir, il avait ensuite distribué son bénéfice à ses concitoyens.

La délibération suivie par Démocrite, telle qu'elle est exposée par Pline, peut être explicitée par les syllogismes suivants, dont nous omettons les mineures par souci de brièveté, sauf pour le dernier (pénultième proposition) :

— Le temps est mauvais lors du lever des Pléiades (6-10 mai).
— Les oliviers bourgeonneront mal.
— Les récoltes d'huile seront mauvaises.
— L'huile sera rare.

48. Voir par exemple *Fondements de la Métaphysique des Mœurs*, Ak. IV, 417, où la volonté est clairement décrite comme une causalité naturelle.
49. (Anscombe [1957] 2000, p. 57).
50. (ibid., p. 84).
51. *Histoire naturelle*, livre XVIII, ch. 68, § 273 (édition des *Belles Lettres* : p. 148). Nous avons trouvé cet exemple chez Jacopo Mazzoni, un humaniste qui traite de la question de la théorie et de la pratique dans son ouvrage de la *Comparaison entre les Philosophies de Platon et d'Aristote* (Mazzoni [1597] 2010).

— L'huile sera plus chère qu'aujourd'hui (car les gens prévoient de bonnes récoltes).
— Celui qui achète de l'huile maintenant pourra la revendre avec profit.
— Je veux m'enrichir.
— J'achète de l'huile maintenant.

Les cinq premiers enchaînements, pris isolément, sont des syllogismes entièrement théoriques. On pourrait les rattacher à des sciences : la météorologie, l'agronomie et l'économie. Seul le dernier enchaînement est un syllogisme pratique au sens traditionnel. Mais ce n'est pas là-dessus que porte l'intelligence pratique, ou en tous cas pas l'essentiel. L'essentiel a été d'organiser, sous l'égide de l'intention principale (s'enrichir) l'ensemble des syllogismes théoriques de manière à aboutir à la majeure du syllogisme pratique (« Celui qui achète de l'huile aujourd'hui, etc. »). L'essentiel a résidé, en quelque sorte, dans la « vivacité d'esprit », selon l'expression d'Aristote, qui a *vu* tous les moyens termes entre cette finalité pratique et le fait observé à l'origine (le lever des Pléiades), les extirpant tour à tour de leurs sciences particulières pour déterminer l'action. C'est en ce sens qu'on doit interpréter l'idée maîtresse d'Anscombe, que l'intention ne doit pas être vue comme la majeure du syllogisme pratique, mais comme son ordre descriptif.

L'autre intérêt que nous trouvons à cet exemple est qu'il montre bien que les raisonnements pratiques ne sont pas limités au schéma séquentiel de la fin et des moyens. Là encore, l'essentiel ne se situe pas dans la détermination de l'ordre *des actions* à réaliser, mais dans la formation du jugement antécédent qui va déterminer l'action unique (acheter de l'huile d'olive) réalisant l'intention. Anscombe parle bien d'un nouvel ordre *descriptif* qui est instauré par l'intention, et non d'un ordre *prescriptif*.

Serait donc pratique toute règle dont la raison d'être se réfère nécessairement à une intention humaine. Ainsi, la *check-list*[52], même si elle consiste seulement en une série de jugements, a pour raison d'être l'intention de décoller. De la même manière, tous les artefacts matériels sont des objets pratiques, car la partie fonctionnelle de leur définition se réfère nécessairement à une intention humaine.

On pourrait objecter à cette vaste expansion du domaine intentionnel qu'une roue d'engrenage, par exemple, se définit plutôt par ses propriétés d'interaction physique avec d'autres pièces, que par la référence à une intention humaine. Cependant, la raison d'être de ces interactions qu'on cherche à susciter doit ultime-

52. Voir plus haut, § 134.

ment être référée à des intentions humaines. Ainsi, Anscombe écrit :

> Une [action] qui implique des interactions avec des artefacts, comme allumer ou éteindre, peut bien sûr être effectuée par un objet inanimé ; mais la description existe seulement parce que nous fabriquons des interrupteurs qu'on peut allumer ou éteindre[53].

En d'autres termes, même si « allumer et éteindre » ne sont pas effectuées par un agent auquel nous attribuons l'intentionnalité, il n'en reste pas moins que notre utilisation de ces termes pour décrire le comportement d'une machine fait référence à une intentionnalité *au second degré*, celle qui nous a conduit de prime abord à fabriquer ces interrupteurs et cette machine. La description de ces derniers « en action » appartient donc à l'ordre intentionnel, ainsi que celle de tous les artefacts et systèmes d'artefacts que nous avons construits, et au sein desquels nous nous mouvons. Ce raisonnement peut être itéré à des artefacts de « troisième intention », comme par exemple des pièces détachées de machines, ou au contraire des systèmes de machines, et ainsi de suite. Un autre exemple pris par Anscombe est celui du téléphone, artefact systémique par excellence, puisqu'au-delà de l'appareil individuel que nous utilisons, il suppose toute une infrastructure sous-jacente de télécommunications, mais également le réseau électrique, le génie civil qui leur est nécessaire, etc.

Une telle position concernant la technique est proche de celle que développe le philosophe de la technique Houkes :

> La technologie est liée à des actions humaines intentionnelles. La plupart des artefacts et processus techniques ne se produisent pas naturellement, mais doivent être conçus et fabriqués. […] Même les artefacts qui fonctionnent plus ou moins automatiquement, comme les alarmes incendie ou les robots de chaîne de montage, nécessitent une surveillance et une maintenance. Parce que la technologie est intimement liée à l'action, il est logique de supposer que la connaissance technologique est intimement liée à l'action de concevoir, d'utiliser, et d'autres actions similaires[54].

Les artefacts sont motivés par les intentions humaines, en tant qu'aides et instruments de l'action. Mais ce faisant, ils motivent des actions de second niveau, comme *concevoir, fabriquer, utiliser* des artefacts, et qui peuvent eux-mêmes susciter des actions et des artefacts d'ordre supérieur. Un concept central pour les philosophes Vermaas et Houkes est celui de *plans d'usage* (*use plans*) qui décrit les emplois possibles d'un artefact. Ceux-ci peuvent outrepasser l'intention origi-

53. (Anscombe [1957] 2000, p. 85).
54. (Houkes 2009, p. 339).

nale du concepteur, comme lorsqu'on utilise une bouteille comme vase à fleurs. Des réseaux superposés d'intentions se développent autour des artefacts, qui sont accompagnés d'une large gamme de nuances prescriptives (usages permis, traditionnels, recommandations, conditions d'usage, garanties, interdictions...) dont la complexité dépasse largement celle de simples procédures linéaires pour accomplir un objectif. Tout un monde socio-technique est ainsi mis en place par l'intention humaine « par-dessus » le monde physique, et qui resterait inconnu au savant qui n'y chercherait que des causes ou des lois naturelles[55].

On voit bien en quoi des ordres descriptifs intentionnels peuvent se superposer sans conflit avec l'ordre descriptif simple. Un charpentier qui ausculte les grands pins d'une forêt afin d'y choisir le mât d'un navire *insère* ces êtres naturels dans un ordre intentionnel sans pour autant interdire au botaniste de les examiner pour de tout autres raisons. De la même manière, des règles théoriques en leur fond, comme celle concernant le bourgeonnement imparfait des oliviers lors d'un printemps tardif, peuvent être mobilisées par un raisonnement pratique et ainsi être vues de manière instrumentale. Il y a donc un dualisme épistémique fondamental, qui distingue la description des choses qui dérive de la simple observation, et celle qui est instaurée par l'intention :

> Ainsi, dans toute opération, nous pouvons réellement parler de deux connaissances : le compte rendu qu'on pourrait donner de ce qu'on faisait, sans recourir à l'observation ; et le récit de ce qui arrive exactement à un moment donné, disons « à la chose » sur laquelle on travaille. L'une est pratique, l'autre spéculative[56].

§ 137. La description pure

Un tel dualisme épistémologique est-il cependant tenable ? Reprenons l'exemple de la machine qui pousse à notre place les interrupteurs. Il n'est pas interdit d'y voir un ordinateur : Anscombe n'a peut-être pas choisi par hasard l'exemple d'actions binaires caractéristiques des composants électroniques. Un programme, quel qu'il soit, du moment qu'il résout un problème, est partie de ce monde intentionnel, et les règles qu'il organise en système doivent être vues comme pratiques. Dans le cas limite d'un programme qui serait écrit

55. (Houkes 2009, p. 341). Voir également Houkes et Vermaas (2004), Houkes, Vermaas et al. (2002) et Vermaas et Houkes (2006).
56. (Anscombe [1957] 2000, p. 88-89).

automatiquement à partir de sa spécification[57], et quand bien même cette dernière serait également générée automatiquement à partir d'un autre programme, on peut appliquer la même réponse d'Anscombe : cette écriture automatique n'aurait pas lieu si, à un certain niveau d'explicitation descriptive, on ne rencontrait pas une intention humaine originaire, un problème posé par un humain qu'il faudrait résoudre.

Si donc les programmes d'ordinateur sont ainsi des systèmes de règles pratiques, que penser alors des règles logiques, qui sont des règles de réécriture et des démonstrations mathématiques ?... De tels objets sont-ils des « faits » ou une « réalité », qui « sont antérieurs [à la connaissance] et dictent ce qui doit être dit », pour reprendre les mots d'Anscombe ? Une telle posture semble relever du platonisme mathématique. On peut également – et telle serait sans doute la posture constructiviste de Kant – les traiter comme des règles régissant des opérations intellectuelles dont la distinction avec des opérations « concrètes » serait toute relative.

En poursuivant cette ligne d'argumentation – contre Aristote et Anscombe – il pourrait paraître étrange d'opposer connaissance par observation et connaissance par intention, comme si les modèles et les théories par lesquels les scientifiques rendaient compte de l'expérience n'étaient pas eux-mêmes des constructions symboliques intentionnelles, soigneusement préparées pour cette tâche (quel que soit le statut syntaxique ou sémantique de leur signification).

Cette opposition semble relever du fait que la connaissance théorique est souvent identifiée au contenu d'une théorie déjà constituée (ses axiomes, ses théorèmes principaux, voire leurs démonstrations et applications principales). Ce contenu est présenté sous une forme non intentionnelle, comme simplement descriptive[58]. On oublie par là ses moments dynamiques, c'est-à-dire, en amont de la théorie elle-même, le moment de sa recherche et de son élaboration, et en aval, le moment de son utilisation et de son application à la résolution de problème.

Le texte d'Anscombe cité[59] suggère une telle position : le savoir théorique serait fondé sur l'observation plutôt que sur l'intention, il chercherait la « conformité aux faits » et il garderait un présupposé « incorrigiblement contemplatif » qui les suppose « antérieurs » à la connaissance. De telles expressions connotent la critique d'une métaphysique implicite plutôt que la délimitation d'un domaine

57. Voir plus haut, § 49.
58. Nous revenons sur cette distinction plus loin, § 138.
59. Voir plus haut, p. 531.

légitime de connaissance. On peut se demander, après notre examen au chapitre précédent de l'histoire complexe du concept de « théorie » si ce n'est pas une telle critique elle-même qui est naïve, et si on peut encore, après Bacon et Kant, réduire ainsi le savoir théorique à l'observation passive des faits.

Anscombe rend bien compte de ce qu'Aristote voulait sans doute dire dans le texte difficile d'*Éth. Nic.* VI.1 lorsqu'il opposait les êtres dont les principes sont nécessaires et ceux « qui peuvent être autrement ». L'ordre intentionnel fait apparaître le mât d'un navire au milieu d'une forêt, une machine dans un assemblage de métaux, un réseau logistique sur une carte géographique, un programme dans une configuration de circuits électroniques. Toutes ces « formes » sont contingentes en ce que leur principe est dans le regard que porte sur elles le sujet. Au contraire, la forêt « elle-même », les métaux « eux-mêmes », les lieux géographiques, les courants électriques – toutes ces « formes » relèvent de la simple observation.

Le problème cependant est de savoir si l'observation peut être libre de toute intention. Le botaniste, le physicien, le géographe, ne posent-ils pas eux-mêmes sur leurs objets des regards particuliers, informés par leurs propres problèmes de recherche, d'ailleurs incompatibles entre eux ? et qu'en est-il des entités théoriques et mathématiques ? Anscombe n'est pas explicite à ce sujet ; elle semble supposer la possibilité d'une description libre de toute intention dans le texte cité, qui rend compte seulement des « faits ». Cependant dans la remarque suivante, qui n'est peut-être qu'incidente, Anscombe semble bien *subordonner* les observations aux intentions, ce qui indiquerait qu'elles appartiennent bien elles-mêmes, comme au second degré, à l'ordre intentionnel :

> Mais le rôle de toutes nos connaissances d'observation dans la connaissance de ce que nous faisons n'est-il pas le même que celui des yeux dans la production d'une écriture réussie ? […] Notre observation n'est qu'une aide, comme les yeux sont une aide pour écrire. Quelqu'un qui n'a pas d'yeux peut continuer à écrire avec un stylo qui n'a plus d'encre, ou ne pas se rendre compte qu'il dépasse le bord du papier […] ; c'est là que les yeux sont utiles ; mais l'essentiel de ce qu'il fait, c'est-à-dire écrire telle ou telle chose, se fait sans les yeux. Donc, sans les yeux, il sait ce qu'il écrit ; mais les yeux l'aident à s'assurer que ce qu'il écrit est effectivement écrit de façon lisible[60].

Dans une perspective kantienne en tous cas, il semble douteux d'affirmer qu'il

60. (Anscombe [1957] 2000, p. 53). Un texte fondamental sur cette question nous semble être *The Intentionality of sensation* (Anscombe 1981), qu'il faudrait étudier en détail pour prolonger ces remarques.

n'y a *nulle orientation préalable* du sujet dans l'acte de perception, puisque celui-ci se construit sur la base des schèmes empiriques dont il dispose. En d'autres termes, il se pourrait qu'*il n'y ait pas*, tout simplement, de savoir descriptif « libre de toute intention », sinon à titre d'*idéal régulateur* de la démarche scientifique. Tout système théorique serait *aussi* un système intentionnel, constitué par les intentions d'une communauté scientifique particulière, partiellement déterminé par sa tradition, ses savoirs accumulés, et orienté par ses problèmes de recherche ; il serait théorique seulement en ce qu'il *viserait à se dégager* de ces prédéterminations et à s'approcher de l'idée d'un système théorique pur et universel, c'est-à-dire désintéressé, selon l'idéal d'Aristote.

La conclusion qu'il nous semble falloir tirer du débat entre ces deux positions, l'une d'inspiration aristotélicienne, et l'autre kantienne, est que la contingence des objets de savoir, ou encore leur subordination à des intentions, ne peut être mobilisée pour déterminer la nature théorique ou pratique de la connaissance qu'à condition de certaines demandes épistémologiques ou même ontologiques fortes, devant lesquelles un philosophe pragmatiste ou sceptique se récuserait sans peine, par exemple que les jugements théoriques ne visent pas d'effets dans le monde.

8.3 Les distinctions modale et finale de la connaissance

Dans cette section, nous étudions d'abord (§ 138) l'argument modal, selon lequel les formes de la connaissance technique et de la connaissance théorique sont radicalement distinctes, l'une ayant trait aux méthodes, et l'autre aux faits. Nous utilisons une analyse récente de Zwart, Franssen et Kroes, des philosophes de la technique, pour dégager le fond logique de cet argument ; nous n'y répondons que de manière provisionnelle. Nous nous intéressons enfin à la troisième distinction entre théorie et pratique indiquée par Thomas d'Aquin, qui concerne la visée de la connaissance. Nous nous appuyons sur les réflexions de philosophes contemporains de la technologie qui utilisent souvent ce critère pour mettre en avant la dynamique propre du savoir technique, caractérisé entre autres par sa localité et son pluralisme épistémique (§ 139). Dans le développement suivant § 140, nous montrons qu'un tel critère reste cependant relatif, ce qui s'observe dans l'entremêlement contemporain des sciences et des techniques : la science a une visée essentiellement ambiguë.

§ 138. L'interprétation modale

La seconde distinction entre connaissance théorique et pratique que fait Aristote est tout simplement qu'elles portent sur des modes d'être différents, l'une sur des délibérations et des méthodes, et l'autre sur des faits. Les délibérations peuvent mobiliser des faits, mais elles ne s'y réduisent pas ; elles traitent du *comment*, et non du *quoi*.

Une telle distinction ne tient déjà plus pour Thomas d'Aquin, qui considère qu'il s'agit là d'une modalité mineure de la distinction entre connaissance théorique et pratique. L'architecte peut considérer *théoriquement* les manières de construire les maisons, c'est-à-dire qu'il peut réduire la méthode à un raisonnement portant sur des faits. Réciproquement, toute connaissance spéculative se fonde de manière ultime sur la mise en rapport d'une fonction cible (à expliquer ou à construire) et d'une structure qui y répond – ce qu'il y a au fond de l'assertion de Thomas que Dieu connaît les choses à la manière d'un artisan.

En d'autres termes encore, les résolutions de problème scientifique et technique n'ont pas lieu d'être *essentiellement* différentes, même si leurs méthodes, leurs critères, leur style et leurs expressions varient. Expliquer les phénomènes, les prédire, ou les contrôler, toutes ces finalités relèvent d'une même posture de résolution de problème, qui est de *concevoir* l'agencement de structures et de processus qu'on sait manipuler (dont on connaît les lois). Que cet agencement soit donné ou qu'il doive être exécuté est contingent à la solution elle-même, ce qui explique qu'elle puisse si aisément être convertie pour servir une fin théorique ou pratique.

On peut cependant juger qu'une telle assimilation est trop rapide. Qu'il y ait convertibilité du savoir pratique en savoir théorique ne veut pas dire qu'ils relèvent du même processus de connaissance ou encore, pour reprendre les termes d'Aristote, de la même vertu intellectuelle. Que la science puisse étudier des constructions techniques avec profit et en tirer des propositions générales ne veut pas dire que le savoir qui a guidé cette construction était dès l'origine constitué de ces propositions. Pareillement, que la science utilise des modes de raisonnements techniques pour expliquer les phénomènes ne veut pas dire que ses résultats soient eux-mêmes des méthodes.

Ce contre-argument est assez vivace en philosophie de la technique contemporaine[61]. Il est parfois rapporté à la distinction ryléenne entre savoir et savoir-

61. Voir plus haut, § 15.

faire[62], et se présente souvent sous forme de la remarque grammaticale que la connaissance théorique relève de la proposition indicative, alors que la connaissance pratique relève de l'ordre impératif. On se rappelle que c'est également sur cet argument que Simon avait fondé l'indépendance des sciences de l'artificiel, en empruntant à Hume sa distinction *is / ought*[63]. On le retrouve aujourd'hui notamment sous la forme de l'opposition du *descriptif* et du *prescriptif*, comme ici chez Walter Vincenti, dans son livre influent sur le savoir de l'ingénieur :

> Le savoir descriptif est [...] le savoir du vrai ou de la vérité ; il est jugé en terme de véracité ou de correction. Le savoir prescriptif est savoir de la procédure ou de l'opération ; il est jugé en terme d'efficacité, ou de taux de succès et d'erreur [64].

Zwart, commentant la définition de Vincenti, précise :

> Le savoir prescriptif consiste en conseils ou prescriptions à propos des actions d'ingénierie à entreprendre pour accomplir un certain but technique. Étant prescriptif, il s'agit d'un savoir normatif concernant la fin et les moyens, il a un contenu empirique, et il peut être explicité en mots. Plus précisément, [ici] le savoir prescriptif d'ingénierie est identifié avec tout savoir concret portant sur des actions d'ingénierie, qu'on pourrait reformuler comme une *norme technique* ayant cette forme canonique :
> (*) Si tu veux accomplir le but technique B et que tu es dans le contexte C, alors tu devrais réaliser l'action d'ingénierie A[65].

La distinction entre savoir prescriptif et savoir descriptif ne correspond pas exactement à celle que nous étudions ici, entre savoir des procédures et savoir des faits. Elle recouvre partiellement la distinction déjà étudiée en nous appuyant sur la réflexion d'Anscombe[66]. Les prescriptions portent en effet sur des intentions (ou des devoirs, etc.) et non sur des faits. Ce n'est pas de cette différence qu'il s'agit ici.

Afin de bien voir la différence, il est utile de s'appuyer sur la formalisation des prescriptions techniques proposée par Zwart, Franssen et Kroes (2018), qui montrent qu'une double transformation est requise pour transformer une proposition simple en une telle forme.

62. Voir plus haut, § 16.
63. Voir plus haut, § 66.
64. (Vincenti 1992, p. 197).
65. (Zwart 2020, p. 5).
66. Voir plus haut, § 136.

La prescription technique prend la forme suivante :

(1) *Si tu veux le but B* et que la situation est C*, alors réalise l'action A*.*
Si tu veux rendre la cabane habitable et qu'elle est froide, alors chauffe-la.

Il semble que la règle « théorique » à son fondement soit la suivante :

(2) *Si C, et A, alors B.*
Si la cabane est froide, et qu'on la chauffe, alors elle est habitable.

À première vue, la différence entre les deux propositions est que la prescription technique 1 enveloppe l'assertion d'une volonté, et sa conséquence un commandement. La logique déontique permet de capturer l'aspect intentionnel de cette prescription. Zwart utilise pour ce faire une logique appelée DDeL (*Dynamic Deontic Logic*), qui permet de formaliser l'idée d'un *transfert* de l'intention ou de l'obligation de voir le but *B** réalisé, à celle de réaliser l'action *A**.

Nous renvoyons à l'article pour le détail de cette formalisation. Il suffit pour notre analyse que son résultat soit le suivant, que Zwart appelle la *norme technique* :

(3) *Si la situation est C*, alors réaliser A* implique que B* advient.*
Si la cabane est froide, alors la chauffer va la rendre habitable.

C'est cette norme technique qui permet de *transmettre les intentions* de la fin vers les moyens, selon l'heureuse expression de Zwart, Franssen et Kroes (2018, p. 22). Cette notion semble capturer parfaitement l'intuition kantienne, que la règle de l'action n'est rien d'autre qu'une règle du jugement, pouvant servir indifféremment à des syllogismes théoriques ou pratiques.

L'analyse de Zwart, Franssen et Kroes ne s'arrête cependant pas là. Ils montrent en effet qu'il y a un écart entre les formulations 2 et 3. La norme technique ne peut être logiquement convertie en 2, comme ils le remarquent avec justesse[67], car si $C*$ et $B*$ peuvent aisément être converties dans des propositions C et B qui décrivent des états de faits (si $B*$ est un objet, on peut dire que la proposition B est « $B*$ existe ») tel n'est pas le cas de $A*$ qui décrit une action à réaliser, donc une transition entre l'état C et l'état B. Si on voulait convertir « réaliser $A*$ » en proposition, il faudrait la mettre à un certain temps (« $A*$ a été réalisée » ou « $A*$

67. (Zwart, Franssen et Kroes 2018, p. 7).

sera réalisée »), qui sont des instances possibles de la proposition « réaliser A* », mais non son sens général.

La solution de Zwart, à laquelle nous souscrivons, est de modéliser d'abord la proposition 3 en logique dynamique propositionnelle (*PDL : Propositional Dynamic Logic*), qui définit les actions A* comme des transitions entre des états S du système considéré, qui sont décrits par des propositions classiques. En reprenant partiellement la notation de la PDL, notre formule 3 s'écrit :

(4) $S \models [\alpha]\, B$

où B représente la proposition « $B*$ existe », S un état initial qui satisfait la proposition de contexte C, et $[\alpha]$ le fait que réaliser l'action α (c'est-à-dire $A*$ dans notre précédente notation) conduit nécessairement à un état qui satisfait B. Intuitivement, les actions α doivent être conçues en PDL comme des chemins ou des segments de chemins qui conduisent d'un état à un autre. Ils peuvent donc se composer séquentiellement ou être des arborescences de chemins.

En d'autres termes, les actions ne sont pas des objets de même ordre que les faits qu'ils parviennent à réaliser. Elles fonctionnent plutôt comme des opérateurs de modalité, nécessitant ou rendant possible certains faits, ainsi que le dit très bien Zwart :

> Alors que la logique des connaissances descriptives est de premier ordre (ou peut-être d'ordre supérieur) avec des distributions statiques de valeurs de vérité, la logique des connaissances prescriptives doit être capturée dans une certaine forme de logique dynamique où les actions peuvent changer la valeur de vérité d'une phrase d'un état du monde à un autre [...] Dans ces logiques, l'écart entre les faits et les valeurs est impossible à combler[68].

Il nous semble (mais cela reste à confirmer) qu'une telle irréductibilité se voit également dans l'impossibilité, en logique de la connaissance, de convertir simplement des propositions concernant le savoir-faire en propositions concernant le savoir descriptif. Dire *« je sais faire X »* ne se traduit pas en *« je sais que [je peux faire X] »*, car cette deuxième proposition ne dit pas qu'on sait *comment* le faire, contrairement à ce qui est entendu par la première. Le savoir-faire n'est pas la combinaison d'un opérateur de connaissance et d'un opérateur de compétence[69].

Il y a donc une logique de l'action dont la différence avec celle des propositions ne réside pas dans l'expression de souhaits et de volontés (ce qui est l'objet

68. (Zwart 2020, p. 114).
69. (Y. Wang 2018, p. 4421), (Herzig 2015, p. 740).

des logiques déontiques), mais dans le fait que l'action revient toujours *à modifier les valeurs de vérité de certaines propositions*, puisqu'elle vise à changer l'état du monde. Cette remarque permet d'éclairer encore mieux la remarque d'Anscombe que, dans le syllogisme pratique, l'intention ne joue pas le rôle d'une majeure déontique (*« je souhaite une vache de Jersey »*) mais qu'elle structure la représentation du monde. Ici nous pouvons préciser que cette représentation comprend aussi bien des faits que des possibilités de *changer* les faits, ce qu'on appelle *actions*. *Voir les choses pratiquement, c'est voir ces possibilités d'action qu'un regard théorique ne sait constater qu'après coup*.

On pourrait aisément sous-estimer la force de l'argument modal, en remarquant qu'il ne change rien au *contenu* de la règle, et qu'il est normal qu'elle joue des rôles différents dans des contextes différents : La proposition *« le feu brûle »* joue certes un rôle différent lorsqu'on observe au loin une forêt sous les flammes et lorsqu'on réfléchit aux moyens de se débarrasser d'une lettre compromettante, mais de telles différences sont gommées dans la résolution de problème, qu'il soit théorique ou pratique, où toutes les propositions sont instrumentales.

Cependant ce n'est pas de cela qu'il s'agit ; un tel argument relève plutôt de la discussion précédente avec Anscombe. La distinction qui apparaît ici est plus élémentaire encore ; elle propose de distinguer *actions* et *états* comme deux domaines de jugements radicalement distincts. On pourrait encore une fois hausser les épaules et montrer qu'il est aisé de convertir états et actions ; par exemple, un organisme peut être vu comme dans un état de repos, mais cet état est en fait l'équilibre de nombreuses *actions* métaboliques ; réciproquement, un processus industriel en pleine action peut être décrit comme un certain *état* souhaitable de l'atelier.

Ces changements de point de vue sont certes possibles, mais pas au sein d'un même problème. Or c'est là que cette dualité est la mieux visible : car il est toujours question, dans un problème donné, de *transformations* de *situations*, et donc de règles distinctes concernant les unes et les autres.

Cette dualité remet-elle cependant en cause la position kantienne ? Il ne nous semble pas. D'abord, comme nous venons de le remarquer, cet argument montre seulement que la distinction entre jugements d'état et jugements d'action est *interne* à chaque résolution de problème, qu'ils sont relatifs l'un à l'autre et en coopération étroite. Aussi la dualité entre leurs objets, *action* et *situation* (ou *état*) ne semble pas pointer vers une grande séparation ontologique entre des domaines

de l'être, mais seulement vers la structure logique de la résolution de problème.

Cela se conçoit mieux lorsqu'on remarque que la PDL est une logique inspirée de la logique dite de Hoare, conçue par ce dernier comme un cadre d'analyse de la correction formelle des programmes informatiques[70]. En logique de Hoare, on écrit directement :

(5) $C\,\{\alpha\}\,B$

pour signifier que si la pré-condition C est satisfaite avant l'exécution de l'action α, alors la post-condition B le sera à son achèvement[71]. Cette logique ne vise donc pas originellement à rendre compte de l'action humaine, mais plus généralement à décrire les règles de l'évolution dans le temps de tout système à partir de ses états et de ses changements élémentaires. En particulier, elle vise de manière centrale des machines logiques opérant dans un temps logique, actées par la succession des jugements du logicien sur son objet. La PDL permet d'expliquer pourquoi, par exemple, si l'on suit tel algorithme de tri, on peut être assuré de parvenir à une série de données bien ordonnée. On voit bien ici que cette dernière notion d'ordre est bien distincte de celle de l'algorithme. De manière encore plus simple, on fait bien la distinction entre un théorème et sa démonstration en mathématiques, puisqu'on peut tout à fait comprendre et admettre la vérité en question, sans pour autant connaître ou comprendre sa démonstration.

Dans ces cas de figure, nous aboutissons à un résultat troublant, que la situation est le jugement lui-même, et l'action l'effectuation d'une inférence. Cela montre que cette dualité est inscrite dans la logique même de l'investigation ; elle est même assez évidente puisqu'il faut *a minima*, pour résoudre un problème, distinguer sa spécification (ce qu'il faut faire) de sa résolution (comme le faire).

Si on admet donc que cette dualité a seulement une valeur logique et locale (propre à chaque problème), alors elle ne s'oppose pas à la doctrine kantienne de l'unité de la connaissance. Cependant, nous ne sommes pas encore en mesure d'étayer complètement cette hypothèse. Par exemple, les distinctions entre un théorème et sa preuve, ou entre un résultat et l'action qui y conduit, nous semblent intuitivement valoir de manière absolue. Nous devons donc prolonger cette analyse, ce que nous ferons à la partie suivante.

70. (Harel, Kozen et Tiuryn 2000 ; Pratt 1976).
71. (Hoare 1969, p. 577).

§ 139. L'horizon borné de la technique

Selon Thomas d'Aquin, on s'en souvient, la finalité de la connaissance est le critère principal par lequel on distingue si une connaissance est théorique ou pratique. Comme un tel critère est extérieur et accidentel au contenu même de cette connaissance, l'unité fondamentale du savoir est ainsi démontrée : « Il est accidentel à un objet saisi par l'intelligence qu'il soit ordonné à l'action ou non[72] ».

La force de cette position se voit d'autant mieux si on affirme, avec Kant, que les concepts sont des systèmes de règles. Prenons l'exemple d'un modèle *proie-prédateur* de dynamique des populations, qui permet de déterminer l'équilibre de deux populations animales, l'une étant prédatrice de l'autre. Ce modèle peut servir à un écologue qui veut rendre compte des fluctuations observées de populations animales, ainsi que cela a été fait dans l'étude des lynx et des lièvres du Canada[73]. Mais il peut également servir à un écologue qui *veut intervenir* sur cet équilibre, dans le cadre d'un projet de réensauvagement, par exemple[74]. *Il utilisera le même programme* pour déterminer les paramètres adéquats de son intervention. En d'autres termes, le même savoir (les équations différentielles et le programme qui les modélise) apparaît comme théorique ou pratique selon le problème qu'il s'agit de résoudre, selon l'intention qui préside à la mobilisation de ce savoir, qu'elle soit d'expliquer ou d'agir.

Cependant on ne retrouve pas tout à fait là l'assertion d'Aristote, qui concède aisément que l'artisan parcourt la même série de causes que le scientifique[75]. Ce qui intéresse le Stagirite est la *dynamique* de constitution du savoir et son horizon. Le technicien arrête son investigation dès que ses intérêts sont satisfaits, ce qui borne son savoir à une connaissance comme accidentelle de la chose avec laquelle il traite. Le scientifique, visant la vraie nature de la chose et dénué de tout autre intérêt, se situe dans une dynamique tout autre.

Kant lui-même semble reconnaître cette distinction essentielle d'intérêt lorsque, au chapitre de la *Méthode Transcendantale* dans la *Critique de la raison pure*, il oppose les systèmes *techniques* de la connaissance dont l'unité est déterminée par des circonstances empiriques, à la visée d'une unité *architectonique* de la connais-

72. *Somme Théologique* I q. 79, a. 11.
73. Voir Finerty (1979) à ce sujet.
74. (Pereira et Navarro 2015, p. 76).
75. Voir plus haut, § 96.

sance, qui serait fondée sur une idée *a priori* :

> Sous le gouvernement de la raison nos connaissances en général ne peuvent former une rhapsodie, elles doivent au contraire former un système, et c'est seulement dans ce système qu'elles peuvent soutenir et favoriser les fins essentielles de la raison. Or, j'entends par système l'unité des diverses connaissances sous une idée. Cette idée est le concept rationnel de la forme d'un tout, en tant que, grâce à ce concept, la sphère du divers aussi bien que la position respective des parties sont déterminées *a priori* . […]
>
> Le schème [du système] qui n'est pas esquissé d'après une idée, c'est-à-dire à partir de la fin capitale de la raison, mais empiriquement, suivant des buts qui se présentent accidentellement (dont on ne peut savoir d'avance le nombre), donne une unité *technique* ; mais celui qui provient d'une idée (où la raison fournit *a priori* les fins et ne les attend pas empiriquement), celui-là fonde une unité *architectonique*. Ce que nous nommons science ne peut naître techniquement, par suite de la similitude du divers ou de l'emploi accidentel de la connaissance *in concreto* à toutes sortes de fins extérieures et arbitraires, mais architectoniquement[76].

Il n'est pas difficile de retrouver ici la vieille opposition aristotélicienne entre le savoir intéressé de la technique et la volonté désintéressée de connaître qui forme l'essence même de la contemplation (*θεωρία*), cependant transposée à un degré supérieur : si la distinction entre utilité et vérité ne peut permettre de séparer les lois théoriques des règles techniques, elle peut néanmoins être observée au niveau de la dynamique même des systèmes de connaissance et de la forme qui en résulte, ce que Kant lui-même semble admettre.

Une telle approche est revenue en force dans les réflexions contemporaines sur la technique, qui cherchent, dans un contexte contemporain, à préciser l'intuition qu'elle constitue un domaine de savoir ayant une dynamique autonome de celle de la science, contre le schéma *applicationniste* issu du positivisme du XIX^e^ siècle, selon lequel le savoir technique serait dérivé de celui de la science. Dans son étude des discussions sur cette question, le philosophe de la technique Houkes donne à cet argument récurrent le nom « intuition TU » (TU pour *Truth* vs *Usefulness*)[77]. Par exemple, Wieringa propose la distinction suivante :

> Les problèmes pratiques appellent à des changements dans le monde, de telle sorte qu'il agrée mieux aux objectifs d'une partie prenante donnée. Les problèmes de connaissance, en revanche, n'appellent pas à changer le monde mais à changer notre

76. *Critique de la raison pure*, A 832-834/B 860-862, Ak. III, 538-539.
77. (Houkes 2009, p. 312).

connaissance du monde[78].

Cette dynamique distincte de constitution du savoir rend les techniques toujours dépendantes de circonstances extérieures, même lorsqu'elles visent à se constituer en système. Il s'agit là de l'idée que nous avions déjà rencontrée dans notre commentaire d'Aristote, à savoir qu'un système technique de connaissance, en tant que tel, n'atteint qu'un degré médiocre d'universalité[79]. Les philosophes contemporains de la technologie précisent cette idée en relevant plusieurs caractéristiques épistémiques par lesquels on différencie les savoirs techniques des savoirs théoriques.

1) La plus importante est celle de *localité*, autrement dit que les concepts, représentations et symbolismes mobilisés par la délibération technique le sont toujours à partir d'un contexte donné, c'est-à-dire d'un problème à résoudre, qui commande leur pertinence, tandis que l'investigation théorique vise à établir des représentations adéquates indépendamment du contexte donné. C'est exactement ce qu'exprime cette remarque de le philosophe de la technique et de l'ingénierie Sjoerd Zwart :

> Les connaissances descriptives sont appréciées en tant qu'elles dépendent le moins possible du contexte. La mécanique de Newton ne produit des résultats concrets que si on lui fournit des conditions initiales précises. Les connaissances prescriptives, en revanche, sont appréciées si elles sont adaptées à un type de contexte exactement circonscrit[80].

Les « conditions initiales précises » évoquent l'artilleur de Kant, qui doit décider de quels paramètres concrets tenir compte, comme les frottements de l'air, pour que le modèle balistique lui permette de prévoir la trajectoire exacte du tir. Cette exigence est contraire à celle de l'exposition théorique, qui privilégie plutôt des problèmes simples, où les lois étudiées s'appliquent directement.

De cette dimension contextuelle et locale de la connaissance pratique découlent plusieurs autres. Nous en citons deux :

2) Elle est *superficielle*, au sens où elle est indifférente aux justifications des savoirs qu'elle mobilise pour la résolution de problème, et se contente qu'ils soient *garantis* (*warranted*). Cette superficialité s'exprime, entre autres, par la méthode de la boîte noire (*black-box*), où l'on s'intéresse aux propriétés utiles de l'objet,

78. (Wieringa 2009, p. 1).
79. Voir plus haut, § 108.
80. (Zwart 2020, p. 9).

mais non à leurs mécanismes sous-jacents, ce qu'exprime ici Bunge :

> Les théories technologiques sont nettement plus pauvres que celles de la science pure ; elles sont invariablement moins profondes, et cela parce que l'homme pratique (*the practical man*), à qui elles sont destinées, s'intéresse surtout aux effets nets qui se produisent et qui sont contrôlables à l'échelle humaine ; il veut savoir comment les choses à sa portée peuvent être mises à son service, plutôt que comment sont réellement les choses en général. Ainsi, l'expert en électronique n'a pas à s'inquiéter des difficultés qui grèvent les théories quantiques des électrons [...] Par conséquent, dès que possible, le chercheur appliqué tentera de schématiser son système comme une boîte noire ; il considèrera de préférence des variables externes (entrée et sortie), et traitera toutes les autres au mieux comme des variables intermédiaires pratiques et sans portée ontologique [...] C'est pourquoi ses simplifications excessives et ses erreurs ne sont pas plus souvent nuisibles, car ses hypothèses sont superficielles[81].

3) La troisième spécificité du savoir pratique découle de la précédente. Un même problème peut ainsi avoir une multiplicité de méthodes de résolution possibles, en fonction de la diversité des « critères des parties prenantes » évoqués par Wieringa. Plus généralement encore, un même objet technique peut être modélisé de plusieurs manières différentes, en fonction du problème à résoudre. C'est ce que Hendricks, Jakobsen et S. A. Pedersen (2000, p. 302) appellent le « pluralisme épistémique » du savoir technique, donnant l'exemple des modèles d'ingénierie civile qui modélisent le béton comme un matériau plastique ou comme un matériau élastique[82]. Pour Hendricks, ce pluralisme implique une dynamique de progrès du savoir technique différente de celle de la science fondamentale, telle qu'elle fut décrite par Kuhn. Tandis que cette dernière progresse de « paradigme en paradigme » en passant par des périodes de remise en cause profondes, la première progresse de manière plus évolutive, car elle est « poly-paradigmatique »[83].

Ces trois spécificités épistémiques des systèmes techniques – localité, superficialité et pluralisme épistémique – découlent de leur dimension contextuelle fondamentale, que la recherche théorique, au contraire, vise à *éliminer*. Ce sont ces spécificités qui permettent de distinguer, empiriquement, les systèmes de savoir théorique et pratique.

81. (Bunge 1966, p. 333).
82. *Élastique* et *plastique* désignent ici des propriétés exactement définie en physique des solides. Voir, pour plus de détail la préface de la thèse de Kostic (2009), p. i.
83. (Hendricks, Jakobsen et S. A. Pedersen 2000, p. 284).

§ 140. Les scientifiques, artisans de la raison

Contre cet argument, que les philosophes contemporains de la technique reprennent à Aristote, Thomas d'Aquin et Kant peuvent rétorquer que la myopie de la technique est elle-même relative aux horizons de ses parties prenantes. Les systèmes techniques peuvent, dans leur intérêt bien compris, emprunter leurs méthodes aux sciences et se fixer pour finalité des problèmes d'ordre supérieur (des problèmes pour résoudre des problèmes). Ainsi, de nombreux jugements théoriques et de nombreux concepts, voire des théories entières, sont développées en vue d'une fin pratique plus ou moins éloignée.

Cela est évident dans toutes les théories appliquées, comme la métallurgie ou l'hydraulique, mais il y a par ailleurs de nombreux systèmes de savoir, qualifiés traditionnellement de scientifiques, dont l'ancrage dans la résolution de problèmes utiles est profond. Ainsi en biologie et en chimie par exemple, on passe continûment de la théorie à la pratique. En mathématiques, la théorie des graphes a son origine dans des problèmes de détermination de « meilleur chemin », mais elle s'est développée ensuite comme une investigation théorique à part entière.

On l'a vu, ce constat de la complémentarité de la théorie et de la pratique est fait très tôt, dans le monde arabe[84]. Lorsque Thomas d'Aquin affirme qu'il est possible d'avoir une connaissance spéculative des savoirs techniques (ainsi l'architecte qui peut considérer les différentes manières de construire la maison, non pour ce faire, mais seulement pour les connaître), il ne fait que constater la réalité des réductions en art dont Vitruve donne le modèle dès l'Antiquité latine. L'art n'est donc pas nécessairement myope, et peut s'engager dans des recherches spéculatives lorsqu'il y voit son intérêt, même s'il est plus lointain. C'est par cet argument du « détour utile » que Bacon fait ainsi l'apologie de la science :

> L'histoire naturelle que nous proposons ne veut pas tant […] être utile par des expériences immédiatement fructueuses, que répandre la lumière sur l'invention des causes […] Car même si les œuvres et la partie active des sciences forment le principal de notre recherche, nous attendons cependant le temps de la moisson, n'ayant aucun penchant pour les récoltes de mousse et de blé vert. Nous savons bien en effet que les axiomes, quand ils ont été correctement inventés, entraînent après eux toute la cohorte des œuvres, et les font surgir non pas ici ou là, mais en marche serrée[85].

Dans ce texte, l'image de la *théorie* comme objet de connaissance constitué de deux

84. Voir plus haut, § 111.
85. Bacon, *Novum Organum* (1620), trad. Malherbe et Pousseur, 1986, p. 83.

phases, l'une de construction, et l'autre d'usage[86], est superposé à l'image économique de l'investissement et du profit, de l'accumulation et de la rente. Ce schéma s'oppose diamétralement à celui promu par Aristote et les théologiens médiévaux, qui comparent la connaissance à une élévation, une libération ou un épurement. Si donc le différentiel de systématicité entre sciences et techniques apparaît comme relatif et accidentel, c'est parce que l'échelle qui permet de le mesurer est elle-même tout entière empruntée au règne de la technique.

Ne se pourrait-il donc pas que notre désir « désintéressé » de connaissance ne soit rien d'autre que la croyance en son utilité à très long terme ? Au niveau institutionnel, la recherche fondamentale se trouve financée aujourd'hui par des institutions publiques et privées professant la croyance que les problèmes parfois les plus abstraits et les plus « gratuits » des mathématiques et des sciences peuvent avoir une utilité extraordinaire seulement quelques années après leur résolution. Cette ambiguïté n'est pas seulement le fait des institutions. À l'échelle individuelle, les motivations des chercheurs sont souvent mixtes, voire difficiles à séparer : on peut être sincèrement intéressé par la connaissance des causes d'un phénomène, et en même temps avoir choisi ce sujet de recherche parmi d'autres parce qu'il a des implications pratiques, voire parce qu'il est bien financé, parce qu'il avancera sa carrière, etc.

Le débat contemporain des technosciences est structuré par ces questions concernant la possibilité de distinguer science et technique et – question qui détermine la précédente – celle de dégager une motivation propre à la science (la volonté « pure » de savoir) de ses déterminations socio-politiques. Il n'est pas nécessaire ici d'entrer dans ce débat[87], car il nous suffit de remarquer, dans un style tout à fait kantien, que cette ambiguïté concernant la *sincérité de notre désir de connaître* ne peut être résolue empiriquement, ni par la psychologie, ni par la sociologie ni même par l'examen des productions de la science (où l'on tenterait par exemple, de distinguer les théorèmes « utiles » de ceux qui seraient « gratuits » ou « beaux »).

Affirmer en effet, ainsi que le fait Wieringa[88], que les changements dans la connaissance ne sont pas aussi des changements dans le monde, semble trahir une conception particulièrement éthérée de sa nature. Cela ne revient-il pas, en effet, à

86. Voir plus haut, § 92.
87. Pour une mise au point utile, voir (Parent 2004).
88. Voir plus haut, p. 545.

compter pour rien les publications des scientifiques, leur impact sur les pratiques des laboratoires, les réorientations des politiques de recherche, mais également leur diffusion dans le public et leur influence subséquente sur la culture, sans même mentionner leurs implications technologiques ? Accepterait-on de dire que tout cela constituerait des conséquences *accidentelles* de l'enquête scientifique, dont la motivation essentielle serait le seul « changement de l'état cognitif » des personnes accédant à cette connaissance ? Sans même souscrire à la philosophie de Dewey[89], on voit ce qu'a de problématique la séparation binaire des effets « mentaux » et des effets « concrets » de l'application d'une règle.

Quant aux particularités épistémiques de la technique évoquées plus haut, on peut les observer aussi dans les sciences. Concernant la « superficialité », un chimiste usera des lois de la physique atomique dans la même indifférence à leurs causes qu'un ingénieur des lois de la mécanique. Le pluralisme épistémique existe bien également en sciences, comme on peut le voir dans la multiplicité des modèles d'une théorie, voire dans la multiplicité de ses *versions*[90], comme pour la mécanique sa formulation newtonienne ou hamiltonienne – sans parler de la survivance, toute pragmatique, de la mécanique classique aux côtés de la théorie de la relativité.

Kant semble avoir été tout à fait conscient du caractère historiquement situé des sciences ; ainsi, juste après avoir défini ce qu'il entendait par *système technique*[91], il qualifie les scientifiques d'*artisans de la raison*, sous-entendant par là que leurs théories sont déterminées partiellement par « des buts qui se présentent accidentellement ».

> Le mathématicien, le physicien, le logicien, quelque éclatant succès que puissent avoir les uns en général dans la connaissance rationnelle et les autres en particulier dans la connaissance philosophique, ne sont toutefois que des artisans de la raison (*Vernunftkünstler*). Il y a encore un maître dans l'idéal, qui les emploie tous, et se sert d'eux comme d'instruments pour avancer les fins essentielles de la raison humaine. C'est celui-là seul que nous devrions appeler le philosophe[92].

La position de Kant est donc que les problèmes que se donnent à eux-mêmes les scientifiques sont partiellement déterminés par leur propre tradition et les conditions socio-culturelles de leur exercice. En cela il n'y a pas solution de continuité

89. Voir plus haut, § 135.
90. (Vorms 2009).
91. Voir plus haut, p. 545.
92. Kant, *Critique de la raison pure*, A 839 / B 867, Ak. III, 542-543.

entre science et technique : toutes deux procèdent de la même faculté de connaître et sont toutes deux entièrement mobilisées à la résolution de problèmes qui leur sont donnés par ailleurs. Cependant la science peut *aussi* être guidée par la raison qui cherche son autonomie en visant l'idée d'une architectonique de la connaissance. Cette idée cependant n'est pas empirique, et elle est l'affaire du philosophe, une figure idéale, immédiatement présentée comme problématique. Aller plus loin requiert un engagement métaphysique concernant la vérité et le rapport du sujet pensant au réel, comme le voit très bien Houkes :

> La différence de buts [entre vérité et utilité] semble présupposer une conception réaliste de la science, selon laquelle les théories scientifiques devraient être interprétées comme des descriptions de (la structure de) la réalité, et la science comme une entreprise continue pour construire des théories plus précises. […] Le large spectre des conceptions réalistes peut être opposé à une autre vision de la science : l'instrumentalisme. Les instrumentalistes cherchent à découpler l'enquête scientifique de la vérité, et à la place insistent sur sa connexion à l'utilité[93].

Ces deux visions, pour Kant, sont également possibles, selon qu'on considère l'exigence de la raison de *postuler* l'intelligibilité du réel, et qu'on interprète les avancées de la science comme des manifestations de cette possibilité, ou selon qu'on cherche simplement à décrire la réalité empirique de la science, en tant qu'elle s'insère dans le réseau social, politique, économique ou psychologique des intentions humaines. Il y a ainsi, comme le dit Peter Dear dans le texte cité[94] en évoquant les images ambiguës, toujours deux manières d'interpréter la finalité de la science.

8.4 La systématicité, entre investigation globale et locale

À la fin de la partie précédente, questionnant la spécificité des méthodes industrielles et de l'informatique relativement à la rationalité pratique telle qu'elle peut être habituellement observée dans les activités humaines, nous avions remarqué que leur proximité revendiquée avec la démarche scientifique semblait prendre sa source dans une commune exigence de systématicité. L'hypothèse a été donc faite que la programmation était l'application d'une attitude systématique à la résolu-

93. (Houkes 2009, p. 318). Voir également (Niiniluoto 1993, p. 3) et (Hendricks, Jakobsen et S. A. Pedersen 2000, p. 288).

94. Voir plus haut, p. 505.

tion de problèmes, qu'ils soient théoriques ou pratiques. Cette hypothèse a motivé les deux questions directrices ont guidé les différentes études de cette partie :

1. En quelle mesure doit-on distinguer les activités théoriques et pratiques de la connaissance ? en particulier, est-il légitime de parler de manière unitaire de *résolution de problème* à travers toutes ces activités ?
2. Que signifient les termes *machine* et *système*, et en quoi permettent-ils l'émergence d'une posture *systématique* de la résolution de problème ?

Nous commençons par prolonger les résultats de la section précédente concernant la première question. Nous avançons la thèse que ce ne sont plus les distinctions aristotéliciennes qui structurent aujourd'hui l'opposition entre connaissance théorique et connaissance pratique, mais la division du travail de l'investigation entre la construction de systèmes (que nous appelons investigation *globale*) et la résolution *locale* de problèmes. Ces deux ordres de l'investigation sont transverses à toute discipline de connaissance, qu'elle soit « théorique » ou « pratique » (§ 141).

Cependant, après avoir récapitulé l'essentiel du dialogue entre Kant et Aristote que nous avons tenté d'organiser dans ce chapitre, nous montrons qu'Aristote pourrait se déclarer à bon droit dubitatif de la perspective kantienne, notamment du fait du silence ou du désintérêt de Kant à propos du processus même de l'investigation qu'il avait au contraire consacré au centre de sa recherche sur la connaissance (§ 142).

Ce silence empêche en effet Kant d'expliquer quelle utilité l'investigation globale (la construction de systèmes) pourrait avoir pour l'investigation locale, et quelle serait donc cette manière *systématique* de résoudre des problèmes, dont nous avons montré qu'elle était en « rupture épistémique » avec ses manières habituelles, pratique ou technique. Nous nous tournons vers les mathématiques, et en particulier la géométrie, afin d'éclairer cette question (§ 143).

§ 141. Investigation globale et locale

Les résultats de la section précédente montrent que le concept d'un système théorique de la nature, qui serait établi indépendamment de toute intention humaine, relève du statut d'une idée régulatrice de la raison. Toute théorie scientifique concrète est « technique », comme le dit Kant, parce qu'elle se développe *à partir* d'un certain état de la recherche, de ses questions, de ses outils conceptuels et technologiques, de ses conditions socio-culturelles enfin. Réciproquement, la planification à long terme des utilités humaines est capable de développer des systèmes

techniques jusqu'à en faire des sciences (quasi-) autonomes, et elle peut même, dans son intérêt bien compris, encourager la recherche « gratuite ». Cela veut dire que les différences concrètes et observables entre les entreprises techniques et les entreprises scientifiques de la connaissance ne sont que des différences de degré sur une échelle continue. Leur seule différence essentielle est logée dans l'œil de l'investigateur, selon qu'il vise à connaître les choses telles qu'elles sont, ou telles qu'il voudrait qu'elles soient.

Mais ici une *autre* articulation au sein de la connaissance se fait jour, qui passe près de l'ancienne distinction aristotélicienne entre théorie et pratique, sans s'y ajuster exactement. Nous l'avions déjà rencontrée au chapitre précédent, à propos de la division du travail entre le savant et l'ingénieur[95], et elle est à bien des égards très élémentaire : car s'il y a d'un côté la résolution de problèmes particuliers, qui peuvent être aussi bien théoriques que pratiques, il y a également cette investigation d'ordre supérieur, qui vise à établir une certaine unité de la connaissance sur un périmètre plus ou moins large, qui critique le savoir acquis relativement à cette visée, fixe les problèmes directeurs de la recherche et les reformule sans cesse en fonction de leurs résultats. Cette investigation peut servir des fins techniques, subordonnées explicitement à la résolution de problème, ou des fins théoriques, dont l'évidence est plus ambiguë. Quel que soit le cas de figure, cette investigation a une réalité indéniable, qu'il faut distinguer de la résolution de problème qui n'a qu'une portée locale. Nous allons appeler une telle investigation *globale*, bien que ce terme soit quelque peu inélégant : les autres termes qui pourraient lui convenir : *architectonique, systématique, théorique* pourraient prêter à confusion dans la tentative d'élucidation qui est ici la nôtre. L'avantage de *global* est qu'il s'oppose directement au caractère *local* de la résolution de problème. L'investigation globale ne résout pas des problèmes particuliers mais, tout comme la méthode ramiste visait à organiser les arguments, s'occupe de mettre en ordre et d'unifier les règles conçues et appliquées par l'entendement. C'est ce que Kant appelle la *raison*.

Les règles empiriques que forme l'entendement à propos de l'expérience, avant qu'elles soient retravaillées par la raison, sont en effet le résultat et le matériau d'investigations locales. Les règles émergent de la confrontation du sujet avec l'expérience, dans laquelle il doit se situer pratiquement et conceptuellement. Cela ne veut pas dire que ces règles sont singulières, puisque justement le travail du juge-

95. Voir plus haut, § 118.

ment réfléchissant est de créer les conditions d'une appréhension cohérente à travers le flux divers de l'expérience. Cependant, elles restent toujours conditionnées à leur contexte et aux problèmes pour lesquelles elles ont été forgées. On peut leur attribuer plus généralement les propriétés du savoir pratique relevées plus haut[96], comme la superficialité de l'analyse causale et le pluralisme épistémique. Par opposition, l'investigation globale de la raison vise à décontextualiser ces règles afin de les élever à une valeur inconditionnelle, au moins idéalement.

L'investigation globale n'est pas dans les faits une recherche séparée de l'investigation locale qu'est la résolution de problème. Elle ne peut en effet progresser que par l'identification de problèmes particuliers qui doivent être résolus l'un après l'autre. Cependant ceux-ci lui sont « ordonnés », tout comme des dérivées locales sont les moments singuliers d'une courbe qui les enveloppe. Toute recherche mobilise des expérimentations, des reformulations d'hypothèses, l'écriture de rapports, des lectures, des présentations à des conférences, etc. – autant de problèmes particuliers et « pratiques » qui sont cependant ordonnés à une investigation théorique d'ordre supérieur. Celui qui demanderait au chercheur *quand* il cherche hors de toutes ces activités poserait une question impertinente, comme le touriste de Ryle qui, après avoir vu les différents collèges et facultés dispersés à travers Oxford, exprimerait le souhait de voir à présent l'*Université*[97].

Ainsi l'explication, qui occupe une si grande importance dans les sciences, n'est pas une activité distincte de la résolution de problème, mais l'une de ses espèces. D'abord il existe aussi des explications techniques, qu'il peut être parfois difficile de distinguer des explications scientifiques, comme on l'a vu. Mais il existe aussi des explications mythologiques, politiques, pratiques, etc. Chacune d'entre elle se réfère à des systèmes de significations différents, et l'investigation peut même bricoler une théorie *ad hoc* lorsqu'elle en a besoin, comme le remarque Ryle. L'explication consiste à *localiser* un phénomène dans un système de significations à l'aide des concepts et des règles d'inférence qu'il lui fournit. Elle est en cela une modalité particulière de la résolution de problème, qui peut se transformer aisément en prédiction et en planification, comme nous l'avons vu.

Nous sommes à présent en mesure de revenir à notre première question directrice, et de justifier qu'il soit possible de parler de manière unitaire de la *résolution de problème*. Ce que nous avons ainsi appelé n'est en effet rien d'autre

96. Voir plus haut, § 139.
97. (Ryle [1949] 2002, p. 16).

que l'investigation rationnelle elle-même, prise dans sa dimension locale, c'est-à-dire occupée d'objets particuliers dans des situations particulières – telle que nous l'avions déjà dégagée dans son unité à la fin de notre lecture d'Aristote[98]. Cela ne veut pas dire que la résolution de problème est *pratique*, au sens aristotélicien, ou concrète (« matérielle »), ou encore occupée d'objets individuels et singuliers. Il peut y avoir des problèmes dans les disciplines « théoriques », comme cela est évident en mathématiques. Des problèmes pratiques peuvent aussi être posés de manière abstraite, comme des problèmes de plus court chemin. Et enfin des problèmes pratiques et concrets peuvent porter sur des situations générales, comme savoir s'il faut toujours prévoir des routes de rechange aux routes principales dans un réseau logistique. Et il existe des problèmes pratiques qui se réduisent au simple exercice du jugement, comme nous l'avons vu avec l'exemple de la *check-list*[99].

Il y a ainsi deux ordres de l'investigation :

(i) L'ordre *local* et élémentaire est la résolution de problème ; on l'appelle *délibération* dans les domaines ayant trait à l'action. Le terme générique que nous emploierons dorénavant de manière synonyme à *résolution de problème* est *investigation*, en référence à la ζήτησις aristotélicienne qui est la recherche du moyen-terme ou de la cause[100]. Nous préciserons qu'il s'agit d'une investigation *locale* lorsqu'une confusion est possible. Dans un registre kantien, cet ordre correspond à l'activité de l'entendement, qui établit et mobilise des règles dans sa confrontation avec l'expérience.

(ii) L'ordre *global* et supérieur de l'investigation est la construction de systèmes à partir des savoirs déjà acquis, c'est-à-dire de l'organisation et de la mise en cohérence des règles les unes relativement aux autres. Cette investigation supérieure se particularise immédiatement en investigations d'ordre inférieur, et finalement donc, en problèmes particuliers comme des assertions à démontrer, des prédictions à vérifier, des explications à donner, mais également des rapports à écrire, des expérimentations à mener, des communications à faire. L'investigation globale est l'activité de la raison kantienne.

Ces deux ordres sont transverses à toute forme de discipline et de champ de savoirs. Il peut y avoir une pratique de la chimie et de l'astronomie, comme nous l'avons vu[101], tout comme il peut y avoir une théorie du piano. Ici pratique et théorie

98. Voir plus haut, § 106.
99. Voir plus haut, § 134.
100. Voir plus haut, § 103.
101. Voir plus haut, § 73.

doivent être entendues dans le sens qui se dégage à partir d'Al-Farabi, comme les deux parties complémentaires de toute discipline de connaissance, l'une occupée de résoudre les problèmes qui se présentent dans son champ, l'autre d'organiser et d'interpréter le savoir acquis. La distinction des fins de la connaissance n'est pas ici essentielle pour notre propos. On peut certes dire que la pratique de l'astronomie est ordonnée à l'augmentation du savoir astronomique, tandis qu'au contraire la théorie du piano n'a d'intérêt que si elle sert, d'une manière plus ou moins directe, sa pratique ; mais de telles assertions pourraient être contestées, et le cas de la chimie, par exemple, est bien plus délicat à trancher. Tous ces débats, en tous les cas, dépassent le cadre de notre travail, et n'affectent pas les structures fondamentales que nous avons ici dégagées.

Pour notre propos, il est plus intéressant de distinguer les champs de savoir selon qu'ils se prêtent à une investigation globale fructueuse, ou non. Ce que nous appelons ici *fructueux* peut être pris au sens de Bacon[102], c'est-à-dire qui *converge vers des principes premiers* dotés d'un pouvoir important quant à la résolution de problème. On se rappelle qu'Aristote avait réservé cette propriété à la connaissance de *ce qui est*. Pour lui, les savoirs orientés vers la réalisations des intentions humaines matérielles étaient empreints d'une contingence telle que toute investigation globale se perdrait dans les méandres de facteurs particuliers toujours plus nombreux[103] ; d'où son idée que la technique n'était capable que d'une universalité pauvre et locale, c'est-à-dire de systèmes « rhapsodiques » comme le dit Kant, qui semble être de la même opinion qu'Aristote à ce sujet. Nous avions tenté de montré cependant que, même dans un cadre aristotélicien, la *construction* de systèmes techniques restait une possibilité de droit[104].

Cependant cette discussion est elle-même dépendante d'une question plus élémentaire encore, qui n'est rien d'autre que la seconde question directrice de notre partie : *Quelle utilité ont donc les systèmes pour la résolution de problème ?* Lorsque nous parlons d'une résolution *systématique* de problème, nous semblons bien supposer qu'une investigation globale préalable a rendu disponible un système de connaissances dont les investigations locales peuvent à présent profiter. Mais il n'est pas du tout clair encore *de quelle manière*, et les idées d'*application* et de *déduction* par lesquelles on décrit souvent cette relation

102. (Bacon [1620] 1986, p. 160).
103. Voir plus haut, § 108.
104. Voir plus haut, § 109.

nous semblent floues et trompeuses.

Sur toutes ces questions, la philosophie kantienne nous semble silencieuse, et même en retrait des intuitions d'Aristote concernant le *Pont aux ânes*. Dans le développement suivant, nous tentons de comprendre pourquoi, en récapitulant les résultats du dialogue que nous avons tenté d'organiser dans ce chapitre entre les deux grands philosophes ; nous revenons à la question de la systématicité dans notre dernier développement.

§ 142. Réponses d'Aristote à Kant

En faisant des règles les objets élémentaires de la réflexion, Kant aplanit de manière décisive le territoire de la connaissance, rendant ainsi visible son unité théorique et pratique : l'entendement, sollicité par la volonté, n'établit pas d'abord des règles pour le jugement, dont il dériverait ensuite des règles pour l'agir, qui en seraient distinctes : car tout cela ne sont que des étapes d'un même raisonnement, effectué par une même faculté. Les règles par lesquelles nous jugeons peuvent devenir impératives pour notre volonté, en ce qu'elles indiquent à celle-ci quelles conditions réaliser, si elle vise leur conséquence. En cela, Kant se situe au terme d'une longue tradition, issue de Jean Philopon, qui affirmait déjà que le contenu de la connaissance restait inchangé lorsque la raison la mobilisait à des fins pratiques. En disant que ce contenu est constitué de *règles*, Kant donne, nous semble-t-il, la raison profonde de cette assertion.

Kant ayant très peu écrit sur la technique, nous nous sommes demandé en quelle mesure ses analyses centrales de la *Critique de la raison pure* concernant les jugements empiriques étaient applicables aux jugements techniques. Nous avons suivi pour ce faire l'interprétation de Béatrice Longuenesse, qui met le concept de *règle* au centre de son interprétation de la table logique des jugements proposée par Kant. Pour Longuenesse, les formes logiques (catégorique, hypothétique ou disjonctive) des jugements reflètent les différentes relations que, dans une règle donnée, une condition peut entretenir avec sa conséquence. Les relations catégorique et hypothétique sont duales l'une de l'autre, en ce qu'elles expriment l'intériorité ou l'extériorité de la condition à sa conséquence. Dans le cas d'un jugement technique, cela signifie de manière assez transparente que l'effet attendu d'une machine peut être produit de manière autonome, sans autre condition (jugement catégorique de l'ingénieur), ou qu'il requiert que soit réalisée une condition spécifique dans la situation (jugement hypothétique de l'ingénieur). Dans notre modèle

contractuel de la programmation, cette dualité traduit la séparation nécessaire des responsabilités entre la situation et la machine, ou encore entre le maître d'ouvrage et le programmeur : la satisfaction des jugements catégoriques est de la responsabilité de l'ingénieur, et celle des jugements hypothétiques de celle du maître d'ouvrage. L'acte même de cette séparation – qui n'est souvent nullement évident, et doit résulter d'une réflexion technique préalable – est le le résultat du jugement disjonctif, qui doit analyser l'ensemble des conditions de la situation relativement aux effets souhaités, afin de déterminer *quelle machine* y insérer : il est l'objet de la négociation entre maître d'ouvrage et ingénieur, où leurs responsabilités respectives se trouvent déterminées.

Ainsi, la machine, insérée dans une situation donnée, peut être l'objet de jugements exacts, qu'ils soient catégoriques ou hypothétiques, parce qu'ils auront été tous été prévus par avance dans un jugement disjonctif (systématique). Réciproquement, *tout concept*, tel que vise à le constituer l'entreprise scientifique, n'est rien d'autre qu'une *machine logique* : un *paquetage de règles*, avions-nous dit initialement, puis un *système de règles*. Mais il doit apparaître à la lectrice à présent que le concept établi scientifiquement est plus précisément encore une *machine*, dans le sens où nous l'avions caractérisée plus haut : comme un lieu *inséré* au sein de la situation afin d'y établir des connexions nécessaires et en réduire l'incertitude. L'entendement scientifique, lorsqu'il constitue l'expérience en synthétisant la matière sensible de l'intuition sous des concepts, ne fait rien d'autre que la peupler de machines inférentielles qui permettent aux jugements de se déployer de manière réglée, et la rendre ainsi prédictible et explicable.

Dans la suite de ce chapitre, nous avons passé en revue les trois distinctions que faisait Aristote entre savoir pratique et théorique, en donnant la parole à leurs avocats post-kantiens. Primauté chez Anscombe de l'intention, qui *génère* des systèmes techniques tout autant irréductibles au monde naturel qu'ils sont contingents ; autonomie logique de l'action relativement aux faits chez Zwart et les logiciens du savoir-faire, caractère parcellaire et pluriel des systèmes de savoirs techniques chez Houkes et les philosophes contemporains de la technologie – tous ces arguments nous ont semblé convaincants et décisifs.

Néanmoins, ces arguments se sont trouvés partiellement neutralisés, et cela, non parce que des arguments plus forts auraient été présentés en faveur de la réduction du savoir pratique au savoir théorique, mais au contraire parce que ce dernier se révélait avoir les mêmes propriétés et les mêmes fondements que le

premier. Ainsi les modèles et les théories scientifiques sont également des objets intentionnels et des artefacts symboliques ; le jugement est lui-même une forme d'action, et la différence de finalité entre les systèmes scientifiques du savoir (la vérité) et les systèmes techniques (l'utilité) ne peut être observée selon des critères empiriques, comme la rigueur ou le degré d'unification ou l'horizon de temps de la recherche.

La question est-elle cependant entièrement tranchée en faveur de Kant ? S'il nous semble qu'Aristote conviendrait de bien des points présentés ci-dessus, il maintiendrait tout de même la distinction entre raison délibérative et raison épistémique.

Aristote accepterait sans doute la distinction que nous avons dégagée entre l'investigation globale (qu'elle soit technique, scientifique ou de toute autre nature) et la résolution locale de problème. Observant l'état actuel du savoir, il accepterait sans difficulté la possibilité de systèmes techniques d'une rigueur et d'une profondeur de vues similaire à celles des sciences. Il admettrait aussi que les théories scientifiques sont provisionnelles, et que l'intellection des principes absolus d'une science (*νοῦς*) est une visée et non un état de l'homme théorique d'*Éth. Nic.* X. Il approuverait l'idée que les délibérations techniques peuvent avoir la rigueur d'une résolution géométrique de problème (puisqu'il en fait lui-même le rapprochement), et plus généralement que toutes les méthodes scientifiques peuvent également être mobilisées par les techniques et les professions, comme il le déclare de celles qu'il a lui-même élaborées dans les *Topiques* et les *Premiers analytiques*. Il accepterait donc aussi, peut-être avec quelque hésitation, l'assertion cruciale qu'il n'y a qu'une seule forme de raison qui délibère, calcule, construit, invente – qui résout toute sorte de problème par la manipulation de règles.

Cependant il jugerait ambiguë l'assertion que les artefacts symboliques produits par les institutions scientifiques (les exposés de théories, les manuels, les articles de recherche) sont empreints du même degré de contingence intentionnelle que les autres réalisations matérielles des personnes humaines. Il exigerait qu'on ajoute qu'elles visent néanmoins à décrire les choses telles qu'elles sont. En cela elles doivent être attribuées à une vertu intellectuelle (*ἐπιστήμη*) qui n'a rien à voir avec la résolution de problème et qui est la capacité à poser les problèmes concernant les choses même, de manière réfléchie et à bon escient : *« qu'est-ce que c'est ? – cela va-t-il arriver ? – pourquoi ? »* etc. C'est cette vertu qui est l'intention et le moteur de l'investigation théorique, et on ne peut la réduire, comme le fait Kant, à

la simple hygiène des règles. L'investigation systématique, telle que l'entend Kant, peut être motivée par des entreprises utilitaires, même si elle prend l'apparence de la recherche pure, et même si dans ses enquêtes elle découvre par accident des lois élémentaires de la nature et de la vie. Tous ces systèmes techniques sont pour Aristote au seul service de la résolution de problème, tandis que les systèmes théoriques sont ordonnés d'abord à la découverte des principes. Les premiers sont des constructions contingentes, tandis que les seconds, tout hypothétiques qu'ils soient, participent néanmoins de la nécessité qui empreint tout discours rationnel visant à décrire ce qui est, et qui reste donc *génériquement* distinct du discours ordonné à montrer tout ce qu'on pourrait en faire.

Cette articulation rend mieux visible le différend entre Kant et Aristote. L'entendement kantien couvre le champ de la raison délibérative aristotélicienne, prise au sens large que nous venons d'indiquer. Résoudre des problèmes en effet, cela n'est rien d'autre que manipuler des règles connues pour interpréter des situations problématiques dans l'expérience, agir en conséquence et, quand ces règles nous font défaut, en former de nouvelles et réorganiser nos systèmes de règles (les concepts) de manière cohérente. La raison kantienne, quant à elle, couvre le champ de la raison théorique aristotélicienne, en ordonnant ces systèmes de règles à des visées d'unification architectoniques sous l'égide de principes. La différence générique qu'Aristote fait entre raison délibérative et théorique n'a plus lieu d'être pour Kant, car il nie que la raison soit une faculté de connaître originale. Bien qu'elle se laisse aller à entretenir une telle songerie, elle n'est rien d'autre que la fonction régulatrice de l'entendement, c'est-à-dire la régulation des règles elles-mêmes, qui veille à les organiser en ensembles cohérents, exhaustifs et minimaux.

Il n'est pas utile à notre réflexion de prendre parti entre ces deux positions métaphysiques concernant la raison théorique. La position minimaliste kantienne nous suffit, puisque nous nous intéressons d'abord à la résolution de problème, et à la construction systématique seulement en ce qu'elle lui est rapportée. Il nous importe donc peu de savoir en quel sens les sciences diffèrent *authentiquement* des techniques, du moment qu'est dégagée l'égale possibilité de leur systématisation : et c'est cela qui nous importe pour comprendre ce que peut être la résolution *systématique* de problème.

Or c'est justement sur ce sujet que, même si la question des prétentions de la raison théorique était mise de côté, Aristote pourrait à bon droit rester insatisfait. Il nous semble en effet qu'il y a une sorte d'*angle mort* dans la philosophie kantienne

de la connaissance, en ce qu'elle traite très peu du procès même de la réflexion, qu'il s'agisse de délibération pratique ou d'investigation théorique. Les hésitations, les explorations, les reprises de la réflexion dans son effort pour parvenir à la connaissance, et donc également les méthodes et les heuristiques, intéressent peu Kant. La *Critique de la raison pure* mentionne plutôt des actes, comme si ceux-ci étaient immédiats, et encore il est significatif que les termes qui les désignent, notamment *synthèse* et *jugement* soient ambivalents, pouvant signifier aussi bien l'acte que son résultat. Cela implique que Kant dit très peu de choses des différentes *manières* de résoudre des problèmes, et notamment de cette *manière systématique* que nous cherchons à élucider.

D'une certaine manière, ce silence est compréhensible. Occupé à mettre en lumière la sphère du transcendantal, qui ne concerne pas le procès de la connaissance mais sa possibilité, Kant veut à tout prix éviter que son entreprise soit confondue avec la psychologie de la connaissance. Il ne s'intéresse pas à la manière dont on parvient à l'idée de *force*, par exemple, mais à la prétention de ce concept d'avoir un sens, de dire ce qui est. Il ne s'intéresse pas aux processus de construction de l'expérience, mais à éviter le scepticisme ou l'idéalisme que cette idée peut naturellement susciter.

C'est cela qui rend le dialogue entre Kant et Aristote difficile. La sensibilité du Stagirite aux formes et aux manifestations de l'investigation intellectuelle et de la délibération pratique, est présente dans toute son œuvre, à commencer par les *Topiques* et la *Rhétorique*, mais également, comme nous l'avons relevé, dans les *Premiers Analytiques* et plus généralement dans toutes les considérations liminaires à ses recherches. C'est l'observation de la réflexion en action, et de ses *vertus*, qui suscite les catégorisations d'*Éth. Nic.* VI ainsi que la distinction entre, d'une part, des relations complexes qu'entretiennent la science (ἐπιστήμη) et l'intellection (νοῦς), et d'autre part celles de la technique (τέχνη) et de l'habileté (δεινότης). Toutes ces notions sont qualifiées par Aristote de *vertus*, c'est-à-dire d'activités de l'esprit qui diffèrent par leurs modalités et leur visée.

La philosophie de Kant est indifférente à ces distinctions. Si elle identifie si aisément connaissance technique et connaissance théorique, c'est qu'elle ne compare que leurs régimes de vérité, et qu'elle n'y trouve qu'une différence de degré et non de nature. Tout occupé à distinguer la sphère du transcendantal de celle de l'expérience, Kant devient indifférent, paradoxalement, à décrire l'activité constructive du sujet, qu'il promeut par ailleurs si fortement. Il semble ainsi reproduire, à un

niveau plus fondamental, le vis-à-vis qu'il dénonce entre sujet et objet, dans l'écran qu'il place entre le sujet transcendantal et les phénomènes de l'expérience, du côté desquels il rejette toutes les descriptions empiriques de l'activité intellectuelle. Il n'a pas échappé à la lectrice que nous avons soigneusement évité d'évoquer le domaine transcendantal. Notre intention n'était pas là de « naturaliser » Kant, mais de rester sur un terrain empirique où le dialogue serait possible avec Aristote, et surtout, où les notions de *résolution de problème*, de *décision*, de *démonstration* et de *construction*, etc. – de *programmation* enfin – pourraient prendre un sens vivant, celui de leur réalité d'*actes* de la réflexion.

§ 143. La systématicité

L'indifférence de Kant au processus même de l'investigation locale est dommageable pour notre enquête, et cela à deux titres. D'une part, elle laisse ouverte une part de doute quant à la réponse à notre première question directrice, concernant l'unité de la résolution de problème à travers les domaines théoriques et pratiques de la connaissance (au sens aristotélicien). Il faut ici revenir à l'argument présenté par Zwart, Franssen et Kroes, qu'une logique de l'action semblerait bien être de *second ordre* relativement aux logiques classiques du jugement. En effet une action, étant la transition d'un état à un autre, *modifie* les valeurs de vérité des jugements décrivant ces états. Nous avons remarqué que cet argument vaudrait quand bien même les « états » en question seraient de simples objets logiques, et si l'action était seulement délibérative. D'un point de vue logique donc, la délibération semble bien être ce qui transforme le jugement lui-même, et dont l'étude est donc de second ordre relativement à celle du jugement.

De facto, les jugements théoriques ne portent-ils pas irréductiblement sur des états de fait ? Nous avons relevé les hésitations d'Anscombe au sujet de la nature des actes d'observation. Et la géométrie et les mathématiques tout entières, que nous prenons pour guide dans l'analyse de la systématicité, ne traitent-elles pas de pures relations entre objets, qu'elles fixent dans des *théorèmes* ?

Nous ne sommes donc pas encore mesure de concevoir l'unité de la connaissance théorique et pratique, ainsi que nous y invite Kant. Reléguer l'étude de la délibération à la psychologie revient à laisser obscure la possibilité d'une *transformation rationnelle* du jugement. On protestera que la logique étudie justement les lois de l'inférence valide et du raisonnement. Cependant ce n'est pas de cela qu'il s'agit ici : car la question est plutôt de savoir si on peut identifier la délibé-

ration au raisonnement ; et l'ensemble de notre réflexion jusqu'ici nous conduit à y répondre négativement. Le raisonnement semble être en mesure de récapituler l'activité de résolution de problème, mais celle-ci, avec ses tours et ses détours, ne semble pas se confondre avec lui ; au contraire c'est, à bien des égards, elle-même qui le *construit*. Au-delà donc de la distinction entre investigation « globale » et investigation « locale », il semble rester une différence modale importante entre jugement pratique et théorique. La perspective kantienne concernant l'unité de la connaissance nous invite à concevoir l'un et l'autre comme deux moments dynamiques d'un unique procès de la connaissance, mais cette possibilité nous est encore obscure.

D'autre part, l'indifférence de Kant à l'investigation nous empêche de comprendre ce que serait la résolution *systématique* de problème, entendue comme *manière spécifique* de l'investigation locale qui se situe dans une certaine « rupture épistémique » avec ses manières pratiques et techniques habituelles, et qui semble s'appuyer sur les acquis de l'investigation globale, sans que nous puissions expliciter comment. Notre tentative de répondre à notre seconde question directrice, qui cherche à comprendre la nature et la possibilité de cette manière de résoudre des problèmes, se trouve ainsi bloquée.

Or ici il nous semble y avoir une régression nette relativement aux réflexions d'Aristote qui, à travers le *Pont aux ânes*, nous a semblé avoir une première intuition de la nature et de la généalogie d'une *manière systématique de résoudre des problèmes*. Si, à strictement parler, cette méthode requiert seulement qu'on lui fournisse un espace de problème qu'elle peut explorer exhaustivement, elle ne devient vraiment systématique que lorsque cet espace est lui-même dérivé ou généré par un système. Le *Pont aux ânes* permet de résoudre rigoureusement un problème, *du moment que* celui-ci est correctement posé. Comme les machines de Jackson[105], il est une garantie locale de résultat qui permet de formuler des demandes globales à son environnement. La résolution systématique d'un problème requiert que son environnement accomplisse sa partie du contrat, celui de s'organiser en système.

Cette intuition n'a pas de postérité évidente aux Temps modernes. Si le *Pont aux ânes* se développe dans le commentaire aristotélicien au Moyen Âge, il tombe dans l'oubli à la Renaissance et cela chez les aristotéliciens eux-mêmes[106]. L'utilité des systèmes pour ce qu'on appelait encore l'*invention* au XVII^e^ siècle n'est pas

105. Voir plus haut, section 3.3.
106. (Hamblin 1976 ; I. Thomas 1965).

perçue par La Ramée qui se réfère plutôt, pour organiser celle-ci, à l'idée d'une *Topique*, plus pragmatique car plus circonstancielle. Les systèmes servent ainsi à accumuler le savoir une fois constitué, mais pas à le générer. Ils peuvent *donner* une solution qui y a été consignée par avance ; on leur demande seulement de les bien classer et cartographier en tables et en arborescences afin que leur recherche soit efficace, de la même manière qu'on demande à une bibliothèque de pouvoir servir efficacement les ouvrages qu'on y a entreposés.

La fidélité à l'esprit du *Pont aux ânes*, et à la croyance en l'utilité des systèmes pour l'invention, se retrouve sans doute le mieux dans la tradition lulliste, à laquelle Alsted et Leibniz participent. Pour ces représentants de ce qu'on a appelé la « rationalité baroque » il doit en effet exister une méthode procédurale universelle pour dériver toutes les vérités d'un nombre fini de notions et d'axiomes, qui forment ensemble comme la *clé* de l'univers. C'est cependant la recherche de ces premiers principes qui concentre les efforts de ces penseurs, plutôt que la réflexion sur l'articulation entre les systèmes de la connaissance et les procédures de la découverte.

Plus généralement, l'idée de systématicité qui se développe au XVIII^e^ siècle (l'esprit systématique et l'esprit de système dont discutent les encyclopédistes) est entièrement entendue comme relative à la construction de systèmes, et non à leur usage. Quand Wolff énumère les avantages d'un esprit systématique, il ne pense pas même à invoquer sa compétence accrue à résoudre les problèmes. Au mieux mentionne-t-il l'utilité des systèmes pour la vérification plus aisée des propositions. Kant n'ajoute rien à ce sujet.

C'est par les mathématiques, et en particulier par la géométrie, introduite par Galilée et Descartes au cœur des méthodes scientifiques, que le modèle adéquat de l'articulation entre système et résolution de problème sera trouvé : le système est d'abord là pour fournir des *éléments de formulation* des problèmes et de leurs méthodes de résolution, et non pour les résoudre directement, sinon à titre d'exemples et d'outils complémentaires. Cela est la tâche de l'investigation qui doit *concevoir* la solution, et qui pour ce faire ne peut pas en général s'appuyer sur des recettes fixes – cela semble être le prix à payer pour disposer, une fois cette conception établie, d'une exécution *exacte et exhaustive* c'est-à-dire, d'une part, sans faille ni approximation, où chaque opération doit avoir été préalablement et précisément définie, et d'autre part, sans exception, valide pour en toute circonstance et en tout lieu d'application de la solution. On n'est plus ici ni dans le schéma arborescent

des humanistes du système se ramifiant par lui-même en direction de tous les problèmes possibles, ni dans le schéma combinatoire des baroques, qui multiplie les tables de symboles au risque de devenir un immense tarot. Au contraire de tout cela, le modèle géométrique de la systématicité favorise, dans la résolution de problème, les divisions strictes du travail qui vont inspirer l'ingénierie militaire du XVII^e siècle.

Cependant pourquoi la géométrie peut-elle représenter tous les problèmes dans son champ, sans pour autant les indexer de manière préalable ? Et pourquoi permet-elle de construire des solutions systématiques, mais cela non systématiquement, c'est-à-dire en appliquant une recette universelle d'analyse des problèmes ? Qu'est-ce qui la différencie, par exemple, des systèmes syllogistiques où une procédure systématique comme le *Pont aux ânes*[107] ou le GPS de Simon[108] sont possibles ? Et pourquoi est-elle plus effective qu'eux ? Y a-t-il, d'ailleurs, d'autres formes de systématicité plus effectives qu'elle ? C'est à ces questions que nous devons à présent nous consacrer.

107. Voir plus haut, § 104.
108. Voir plus haut, § 64, et annexe A.

Quatrième partie

La résolution de problème

Chapitre 9

Construire et prouver (Martin-Löf)

§ 144. Introduction de la quatrième partie

À la fin de notre seconde partie, nous avions insisté sur la continuité des développements opérés par la révolution industrielle et ceux opérés par la révolution numérique : l'emploi des machines pour régler les affaires humaines nous avait semblé procéder d'une même attitude de réflexion *systématique* concernant l'action et la délibération. Cet esprit systématique, qui caractérise aussi bien « les plans [que] les théories », selon l'expression de Ryle, nous était apparu pouvoir expliquer les succès parallèles et de plus en plus interdépendants du scientifique et de l'ingénieur, ces deux nouvelles figures des Temps modernes, succès dont l'aspect *cumulatif* n'est pas le moindre. La partie précédente a tenté de revenir à la source de cette systématicité et d'expliquer son égale applicabilité à la réflexion théorique et à la réflexion pratique : caractérisé initialement par Aristote comme consacré à la science, on voit aux Temps modernes l'esprit systématique tenter de s'appliquer à toutes les pratiques humaines, en reprenant d'abord le schéma des divisions platoniciennes, puis celui de la combinatoire lullienne, jusqu'à se convaincre que son modèle devait s'inspirer de la géométrie, suivant en cela la voie qui commençait d'être tracée par la révolution scientifique.

Ce qu'exige en particulier une approche géométrique de la pratique est de bien définir et normer les actions, de sorte que l'ingénieur puisse concevoir à l'avance la constructibilité de l'ouvrage ; nous avons montré ainsi, en suivant Hélène Vérin, comment une division multiple du travail put se mettre en place, dont le pivot central fut celle de la conception et de l'exécution. Sur ce point exactement, la notion géométrique de *machine* nous a semblé jouer un rôle central, en tant qu'elle

était l'objet de la *construction d'automatismes*, qui réduisait à néant la part du jugement dans l'exécution pour l'enrichir d'autant dans la conception.

Kant mène à son terme la conception de l'unité de la connaissance dans la résolution de problème, en définissant l'entendement comme le *pouvoir des règles*, par lesquelles celui-ci donne tout autant un sens à son expérience qu'il se détermine à agir. Nous avons tenté de montrer que ces deux fonctions de l'entendement, théorique et technique, ne se distinguaient pas tant par leur visée intentionnelle – puisque celle-ci devait toujours s'inscrire dans un champ de problèmes donné, qu'il soit scientifique ou pratique – que par la *raison* qui structurait ces champs, selon qu'elle était motivée par des fins externes d'utilité ou par une fin interne de mise en ordre des règles constituées dans l'expérience. La raison, comme faculté supérieure de l'esprit humain, ne peut en effet se contenter d'un ensemble disparate de règles. Elle exige sans relâche d'unifier ces « systèmes techniques » de la connaissance dans une perspective de plus en plus globale.

La perspective kantienne nous a paru néanmoins incomplète. D'abord, elle occulte, ou tout au moins passe sous silence, le processus même de la délibération, le résumant en général à l'acte terminal du jugement, sauf en de rares occurrences, comme celui où Kant évoque les pro- et les épi-syllogismes de l'entendement[1]. Or un tel silence n'est pas seulement dommageable pour la psychologie de la connaissance, il laisse ouvert un problème logique. Comme nous l'avons vu en effet[2], l'action *transforme les faits*, comme la délibération *transforme le jugement* et ces transformations semblent relever d'une logique d'ordre supérieur relativement à la logique simple des propositions, et donc de règles différentes.

La force de ces difficultés peut être vue assez intuitivement. Il n'est pas nécessaire que, parce qu'en abaissant le levier j'ai mis en marche la machine, qu'elle s'arrête si je le relève. La logique des faits, ici, n'est pas congruente à la logique de l'action. De la même manière, il ne m'est plus possible, après une opération de tri, de *recomposer* par une procédure inverse la série désordonnée de nombres que je considérais auparavant, car une telle procédure n'existe pas. Ici, l'investigation n'est pas réversible, tandis que le jugement, par le biais de la mémoire, l'est tout à fait.

Cette difficulté encourage l'idée que les règles par lesquelles on peut juger d'une situation diffèrent génériquement de celles de l'action (qui transforme les situa-

1. Voir plus haut, § 134.
2. Voir plus haut, § 138.

tions), et de celles de l'investigation (qui transforme le jugement). Or une telle distinction empêcherait de concevoir la notion de *résolution de problème* de manière unifiée à travers tous les domaines de la connaissance, qu'ils soient « théoriques » ou « pratiques ». Surtout, elle remettrait en cause notre hypothèse, à savoir que l'approche industrielle et informatique de la résolution de problème se caractérise par la *même posture systématique* que celle qu'on trouve en mathématiques et en sciences.

Or cette difficulté est sensible dans les mathématiques elles-mêmes, puisqu'on y trouve également des actes de construction, de preuve, de calcul, dont on peut se demander quel rapport logique ils entretiennent avec les jugements (théorèmes, résultats) qu'ils produisent. Une certaine tradition classique, sans doute encore majoritaire, incite à concevoir les mathématiques comme « pures » et « théoriques », occupées seulement à *constater* des relations entre entités abstraites, les preuves n'étant elles-mêmes que le dépliage méticuleux de telles relations. Autrement dit, la part d'*investigation* et d'*action* devrait idéalement s'y réduire à néant.

Par ailleurs, les indications de Kant sur la notion de *système* sont insuffisantes. Sa position résume très bien la riche dialectique qui se déploie tout au long du XVIII^e^ siècle à son propos : si toute prétention de la raison à déclarer achevé un système de connaissances doit être combattue, l'entreprise systématique elle-même est néanmoins congénitale à la raison, et le moteur du progrès de nos connaissances. Cependant dans cette perspective qui reste fondamentalement ramiste, le rôle de la systématicité est simplement « administratif ». Il se contente d'organiser le savoir déjà acquis et validé par d'autres moyens. Au contraire, nos analyses du chapitre 7 sur les mathématiques, et en particulier sur la géométrie, nous invitent à concevoir la systématicité comme proprement productrice de savoir[3]. Un système est un espace qui procure à l'investigation des ressources et des règles pour exprimer des problèmes d'une complexité indéfinie aussi bien que pour les résoudre ; et c'est cette division du travail entre l'investigation systématique globale et la résolution locale de problème qui, semble-t-il, créée la puissance *générative* et explosive des théories modernes[4]. Leur indifférence aux problèmes, que certains diront feinte, est la raison même de leur succès.

Si cette hypothèse est exacte, on perçoit alors l'affinité profonde qui pourrait exister entre les systèmes de connaissance et ce que nous avons appelé *systèmes*

3. Voir plus haut, § 116.
4. Voir plus haut, § 118.

objectifs, principalement les institutions socio-politiques et les ensembles techniques[5], si on les considère également comme des espaces de ressources et de règles. Cependant, il n'est pas clair si un tel rapprochement est simplement métaphorique ou s'il fait signe vers une liaison plus profonde entre ces objets. Dans le second cas, on pourrait localiser la parenté profonde des révolutions scientifique, industrielle et numérique dans ce qu'on pourrait appeler l'*invention de la systématicité*, c'est-à-dire cette idée que la résolution de problème, de quelque nature qu'elle soit, est bien plus effective lorsqu'elle peut se déployer dans des cadres préparés et contrôlés à l'avance – des théories lorsqu'il s'agit d'entreprises de connaissance, et des espaces techniques ou institutionnels normalisés (l'usine, l'administration) lorsqu'il s'agit d'entreprises productives ou socio-économiques. La nature de la programmation de machines pourrait alors apparaître bien plus clairement.

Les deux questions directrices de la partie précédente sont donc reconduites ici encore, quoiqu'elles soient plus précises à présent :

1. Comment comprendre, qu'en tout problème, une dualité semble instituée entre action et situation, entre états et transitions d'état ? Faut-il la comprendre comme relevant de catégories plus générales de l'intelligibilité des choses, ou (ce qui est l'hypothèse que nous allons défendre) comme *interne* à la logique de la résolution de problème ? En particulier, comment concevoir la relation entre le jugement et la *transformation* du jugement dans l'investigation ?
2. D'autre part, comment est-il possible, en mathématiques, de concevoir des systèmes organisés de règles qui, tout en étant finis, voire minimaux, permettent d'exprimer la richesse indéfinie des problèmes de la connaissance et de l'action, et de donner les outils pour les résoudre ? On se rappelle qu'un des problèmes de la syllogistique aristotélicienne était qu'elle ne permettait pas d'expliquer que des théories pussent générer des théorèmes en nombre indéfini[6], et que ce caractère clos avait pu jouer un rôle dans le scepticisme d'Aristote quant à l'expansion possible de la technique hors de l'artisanat traditionnel.

Dans cette partie, nous allons tenter de répondre à ces questions en examinant deux théories de la connaissance qui ont également placé la résolution de problème au fondement de l'activité de l'esprit. Dans ce chapitre, centré sur les mathéma-

5. Voir plus haut, § 126.
6. Voir plus haut, § 107.

tiques, nous étudions la théorie constructive des types élaborée par le logicien et mathématicien Per Martin-Löf, qui est aussi bien une proposition de fondation des mathématiques qu'une interprétation de leur nature, ancrée dans l'intuitionnisme. Le chapitre suivant est consacré à la théorie de la connaissance de John Dewey, qu'il appelle lui-même *logique* et *théorie de l'investigation* dans sa somme éponyme de 1938, et qui va fournir un cadre épistémologique puissant pour nos questions.

Le choix de Martin-Löf et Dewey comme interlocuteurs privilégiés de ce chapitre et du suivant, respectivement, a été dicté par le caractère central qu'occupent ces questions dans leur philosophie. Dans une certaine mesure, ce sont eux-mêmes qui nous ont guidé jusqu'à ce point de notre réflexion où nous sommes capables de les formuler ainsi. Il est par ailleurs naturel que nous nous tournions vers l'intuitionnisme mathématique et le pragmatisme philosophique. D'une part, l'intuitionnisme est, à bien des égards, la logique naturelle de l'informatique[7], et la théorie constructive des types de Martin-Löf y trouve une place de choix. Il peut nous permettre d'éclairer une question qui nous suit depuis l'introduction, quant aux raisons de l'association native de l'informatique et des mathématiques. D'autre part, les pragmatistes américains ont exploré parmi les premiers la nature énactive de la connaissance et sa continuité avec l'action humaine, elle-même tout empreinte de logique. Il aurait été possible, et peut-être plus naturel, d'interroger Peirce et C.I. Lewis sur ces questions de logique que John Dewey ; nous n'avons pas de justification forte à ce choix, à part que la pensée de Dewey nous était simplement mieux connue, et se trouvait répondre assez directement à nos questionnements. Peut-être est-ce dû au fait que la logique n'est pas un point de départ naturel pour Dewey mais un point d'arrivée. Il a dû difficilement se convaincre de sa pertinence pour résoudre les questions humaines, ce qui consistait pour lui à l'ancrer dans le terreau du sens commun, qui émerge lui-même de la vie sociale et technique.

La convocation de ces deux réflexions ne nous semble pas exclusive d'autres interprétations des mathématiques (par exemple structuralisme, formalisme, platonisme) et de la pensée rationnelle en général (réalisme, idéalisme, scepticisme, etc.). Nous cherchons à rendre nos réflexions *peu engageantes* philosophiquement, c'est-à-dire à nous assurer qu'elles ne requièrent pas l'assentiment à des principes ontologiques ou épistémologiques forts concernant la nature du sujet, du réel, et de la signification. Plus précisément, nos réflexions se situent sur le simple plan

7. (Dubucs 2008, p. 51), (Fehlmann et Kranich 2020).

descriptif d'une certaine forme d'intentionnalité, ou encore d'une logique, au sens où Dewey donne à ce terme. Ce plan intermédiaire d'interprétation ne prétend ni décrire les processus cognitifs eux-mêmes, ainsi que la psychologie ou les sciences cognitives pourraient le faire, ni dévoiler l'essence de la pensée ou les critères de la vérité, ainsi que la philosophie peut le rechercher, mais seulement expliciter ce que le sujet agissant, délibérant et jugeant peut *vouloir faire et dire*, si on suppose seulement qu'il tente d'éviter de se contredire lui-même dans ses intentions. Il ne faut pas oublier que l'ensemble des termes que nous considérons ici – règles, situations, problèmes, actions, méthodes, procédures, machines, etc. – sont des termes intentionnels et, qu'à ce titre, nous cherchons seulement à articuler la logique de ces intentions. Cette remarque permet d'ailleurs d'entrevoir la possibilité d'une approche phénoménologique de notre sujet, que nous remettons soigneusement à des réflexions ultérieures.

§ 145. Introduction du chapitre

Ce chapitre vise à aborder les questions évoquées ci-dessus dans un cadre mathématique. Les mathématiques, dont les objets et les actes sont réputés « purement intellectuels », laissent apparaître dans toute son acuité la question de la relation entre le jugement qui porte sur des états de fait ou sur des relations, et le jugement qui *décide* comment transformer ces états de fait. D'un côté, comme nous l'avons remarqué ci-dessus, la tradition principale d'interprétation des mathématiques, depuis la géométrie grecque jusqu'à la théorie des ensembles contemporaine, les voit comme tout occupées à découvrir et à établir des théorèmes concernant de purs objets relationnels. D'un autre côté, les mathématiques sont également des pratiques dont les actes élémentaires sont eux-mêmes objets d'une attention particulière : constructions, calculs, preuves, qui permettent de transformer des figures, de trouver des résultats, d'établir des vérités. Cette interprétation pratique, que Platon déplore déjà[8], est à l'origine d'une controverse ancienne, rapportée par Proclus, concernant la géométrie et la primauté qu'on doit y accorder aux théorèmes (qui établissent des vérités) ou aux problèmes (qui décrivent des méthodes de construction de figures). Elle semble en tous cas avaliser un dualisme des jugements « théoriques » et « constructifs » qui nous semble incompatible avec la doctrine kantienne de l'unité de la connaissance. Celle-ci exige plutôt de concevoir le jugement et la transformation du jugement comme deux termes complémentaires,

8. Voir plus loin, p. 593.

présents au sein même de tout acte mathématique atomique. Or c'est justement cette unité que tente de concevoir la théorie constructive des types.

La dimension *constructive* de la géométrie peut également éclairer son caractère génératif. Elle est l'objet des recherches actives des logiciens contemporains. Jean-Yves Girard a ainsi développé une *géométrie de l'interaction* pour obtenir une définition mathématique de la notion d'algorithme ; c'est dans le prolongement de ces recherches que se situe le programme ANR « Geometry of Algorithms » piloté par Alberto Naibo[9], dont l'ambition vise l'articulation précise des concepts de *machine* et d'*algorithme* – ce qui rejoint notre propre questionnement concernant la *géométrie des machines*[10]. En théorie homotopique des types (*Homotopy Type Theory*, dont l'acronyme courant est « HoTT »), l'inspiration géométrique est également explicite, dans le sens constructif que nous venons d'indiquer, comme l'exprime l'ouvrage de référence présentant cette théorie (qui appelle *synthétique* ce que nous avons nommé *constructif*) :

> La théorie homotopique des types fournit une description synthétique des espaces, dans le sens suivant. La géométrie synthétique est une géométrie dans le style d'Euclide : on part de certaines notions de base (points et lignes), de constructions (une ligne reliant deux points quelconques) et d'axiomes (tous les angles droits sont égaux), et on en déduit logiquement les conséquences. Cela est en contraste avec la géométrie analytique, où des notions telles que les points et les lignes sont représentées concrètement à l'aide de coordonnées cartésiennes dans $\mathbb{R}^n$ – les lignes sont des ensembles de points – et où les constructions et axiomes de base sont dérivés de cette représentation.

Tous ces projets sont ancrés dans la tradition de la logique intuitionniste qui, dès les années 1930, proposa de considérer les notions de *problèmes* et d'*actes de connaissance* comme fondamentales en mathématiques, et qui trouva dans les années 1970 un regain d'intérêt important à travers la théorie constructive des types, dont une figure majeure est le logicien et philosophe Per Martin-Löf. Explorer ces projets très techniques entraînerait trop loin cette réflexion. Par contre, mettre en regard géométrie classique et théorie des types, le plus simplement possible, promet de dégager une idée générale de la constructivité, ancrée dans la pratique originelle des mathématiques mais également rendue exacte par les progrès récents de la logique. Cette idée n'est rien d'autre que celle que nous dont nous explorons la possibilité depuis le début de cette réflexion, et que nous avons diversement

9. Voir plus haut, § 21.
10. Voir plus haut, § 124.

nommée jusqu'ici : « science des opérations », « mathématiques » ou « logique » de l'action[11], et que les mathématiciens et les logiciens contemporains se sont employés à rendre exacte.

En tant qu'ils font œuvre de mathématiques ou de logique (et non de psychologie, de philosophie ou d'histoire) – les mathématiciens ne nous parlent cependant de l'investigation *qu'une fois celle-ci achevée*. Leurs preuves et leurs constructions disent à l'investigateur ce qu'il faut faire pour réaliser un jugement ou construire une figure, mais ils ne lui disent pas *comment ils l'ont eux-mêmes découvert*. Cette investigation qui efface ses propres traces nous montre la puissance de la systématicité, mais non comment elle fut constituée. Cela signifie, dans les termes de notre réflexion, que les mathématiciens nous donnent le programme, mais ne nous disent rien de la programmation elle-même. Ils nous parlent toujours de raisonnement, mais non de délibération. Ce chapitre se tourne donc également vers la philosophie des pratiques mathématiques, un mouvement qui s'est développé à partir des années 1980 en réaction à la réduction de la réflexion méta-mathématique à la seule question des fondements[12]. Ce mouvement s'interroge en particulier sur la signification de l'activité mathématique en tant que telle, notamment dans sa relation aux autres activités rationnelles humaines.

Ce sont donc ces deux lectures contemporaines, constructive et pragmatiste (au sens large d'une épistémologie qui accorde à la notion de *pratique* ou d'*action* un rôle fondationnel ou central) de la géométrie et des mathématiques, qui vont nous accompagner au cours de ce chapitre. Notre première section 9.1 vise à les introduire succinctement en prenant pour fil directeur la relation originale entre *problème* et *preuve* qu'elles tentent de mettre au cœur de l'activité mathématique. Nous dégageons de cette étude une première réponse à notre question concernant la relation entre le jugement et la transformation du jugement.

Nous mettons ensuite cette interprétation (section 9.2) à l'épreuve de la géométrie euclidienne, où cette question est l'objet d'une ancienne controverse, rapportée par Proclus, sur la primauté des *théorèmes* ou des *problèmes*. Nous suivons en particulier l'application que font Mäenpää et Von Plato, deux élèves de Martin-Löf, de la théorie des types à la géométrie euclidienne, et nous la jugeons partiellement insatisfaisante. La section 9.3 suggère des pistes pour une nouvelle interprétation

11. Voir par exemple § 62, § 87.
12. Voir notamment l'anthologie de (Tymoczko [1986] 1998) qui reprend les principaux articles historiques de ce mouvement. Plus récemment, voir (Lolli, Panza et Venturi 2015 ; Mancosu 2008 ; Van Kerkhove 2009).

constructive de la géométrie, qui nous permet de corriger assez fortement notre compréhension de la dualité des problèmes et des théorèmes en géométrie. À la section 9.4 nous pouvons dégager enfin le sens et la possibilité d'une approche *systématique* de l'investigation, et esquisser par conséquent les contours de cette « science des opérations » que nous recherchons depuis le début de cette réflexion.

Ces conclusions ne sont cependant que partielles, car si elles nous donnent une idée exacte de cette théorie mathématique de l'investigation, elles ne nous disent rien de sa pratique elle-même, comme nous venons de le remarquer. Notre dernière section 9.5 est consacrée à l'analyse de cet « angle mort » qu'inclut cette perspective, qui procède du fait que les mathématiciens *effacent leurs propres traces* dans leur travail d'investigation. Il s'agit là d'un fait général et central de l'investigation : la méthode qu'elle dégage pour résoudre un problème n'a souvent qu'une ressemblance lointaine avec son propre travail, fait d'hésitations, de retours en arrières et de fausses pistes. Ce fait, apparemment anodin, est la trace qu'il nous faut suivre pour élucider la rupture épistémique qui se produit entre les manières habituelles de résoudre des problèmes et cette posture systématique que nous tentons de dégager. Nous y consacrons le chapitre suivant, où nous suivrons le philosophe qui a consacré à l'analyse de l'investigation les efforts les plus conséquents, John Dewey.

Quelques remarques doivent être faites à l'orée de ce chapitre assez technique, où nous mobilisons, du mieux que nous pouvons, des travaux contemporains de logique et de mathématiques. D'abord nous ne distinguons pas nettement la frontière disciplinaire entre ces deux domaines, puisque notre intérêt porte sur des sujets de fondement des mathématiques qui les entremêlent étroitement ; aussi se peut-il que nous employons l'un où l'autre terme, selon le contexte, sans que cela doive porter à conséquence. Ensuite, nous ne prétendons pas couvrir l'ensemble des travaux pertinents pour notre question, ni désigner « la » bonne logique ou « la » bonne manière de faire les mathématiques. Notre enjeu est de comprendre ce qu'est la résolution de problème, et notre réflexion peut aujourd'hui profiter des travaux extraordinaires qui se déroulent pratiquement sous nos yeux sur cette question – grâce notamment à l'informatique qui permet aux mathématiciens d'accéder à une réflexivité nouvelle sur leur propre activité – sur le travail de la preuve. Ce faisant, notre point de vue est partiel et sans doute simplificateur. Nous ne prétendons pas résumer ces travaux si riches, ni intervenir dans tous les débats qu'ils suscitent, mais interroger certains d'entre eux sur des questions précises qui sur-

viennent à l'occasion de notre propre réflexion. Nous n'évoquons pas, en particulier, les logiques dialogiques et ludiques, qui pourraient éclairer notre questionnement de nouvelles perspectives. Enfin, nous nous écartons, lorsque cela nous semble pertinent, de l'épistémologie sous-jacente des travaux de logique étudiés pour proposer nos propres interprétations. Nous laissons donc dans ce chapitre de nombreuses portes ouvertes, que nous espérons explorer dans des recherches futures.

9.1 Les problèmes mathématiques

Dans cette section, nous commençons par examiner une thèse soutenue avec force par le mathématicien Reuben Hersh selon laquelle les mathématiques auraient affaire essentiellement à la résolution de problème. Sa conception remet en cause la distinction classique entre les problèmes où il s'agit de trouver un objet (par exemple un nombre satisfaisant une équation) et ceux où il s'agit de prouver une proposition. Pour lui, la preuve est simplement ce qui accompagne la résolution d'un problème, dont certains consistent à chercher la vérité ou la fausseté d'une proposition (§ 146). Nous développons cette idée par l'exemple de quelques problèmes simples, ce qui nous conduit à reconnaître trois formes essentielles de résolution de problème (§ 147). Nous les mettons en relation avec les formes de jugement reconnues par la théorie constructive des types élaborée par Per Martin-Löf, dont nous exposons ensuite quelques thèses centrales pour notre propos. Pour Martin-Löf, la preuve n'est rien d'autre que l'acte de résoudre un problème, qui s'achève dans un jugement ; proposition et problème sont ainsi deux termes équivalents, et le jugement est donc aussi bien l'assertion d'une proposition de fait que le procès tout entier qui y conduit – ce qui semble répondre de manière directe à notre question concernant l'identité des règles du jugement et de la transformation du jugement (§ 148). Cette équivalence se voit dans la notion de *fonction* qui peut être interprétée en mathématiques aussi bien comme une transformation dynamique d'un objet en un autre que comme une relation. Si la conception relationnelle (celle de la théorie des ensembles) prévaut en général sur leur conception dynamique, c'est que les mathématiciens ne se préoccupent pas des actes de connaissance eux-mêmes, mais de leurs règles, qu'ils prennent pour objet d'étude et qu'ils étudient dans leur interdépendance (§ 149).

§ 146. La primauté des problèmes en mathématiques (Hersh)

Au début d'un de ses ouvrages didactiques, Reuben Hersh, qui se réclame de la philosophie des pratiques mathématiques[13], s'élève contre la vision populaire qui les assimile à des systèmes déductifs axiomatiques enchaînant théorèmes et preuves. Après avoir conduit le lecteur à travers la résolution d'un petit problème, il écrit :

> Vous venez de résoudre le problème de l'hypercube. Mais vous ne l'avez pas résolu seul : on vous a d'abord posé le problème. Vous avez ensuite reçu des conseils utiles et des encouragements au fur et à mesure que vous progressiez. Lorsque vous avez enfin trouvé la réponse, vous avez reçu la confirmation que votre réponse était juste.
>
> Croyez-le ou non, un mathématicien a des besoins similaires aux vôtres. Il a besoin de découvrir un problème lié à la culture mathématique existante. Ensuite, il a besoin d'être rassuré et encouragé lorsqu'il se débat dans ce problème. Enfin, lorsqu'il propose une solution, il a besoin d'être approuvé ou critiqué. […]
>
> Quiconque a fait des mathématiques sait ce qui vient en premier : un problème. Les mathématiques sont un vaste réseau de problèmes et de solutions interconnectés[14].

Les théories ne sont pas d'abord des systèmes axiomatiques, mais ce champ de problèmes lui-même, qui est évolutif et qui est le fruit des efforts communs des mathématiciens. Les mathématiques progressent de deux manières, d'une part par des résolutions de problèmes qui parfois dégagent de nouvelles pistes de recherches, mais également par l'invention de nouvelles théories qui sont certes en partie déterminées par leur contexte, mais dont chacune est aussi « une création libre de son inventeur [où] nous percevons un saut intellectuel, comme dans un grand roman ou une grande symphonie[15] ».

Hersh prend *problème* au sens très général qu'il a en mathématiques contemporaines. À la lecture de son ouvrage, on comprend qu'une solution d'un problème peut être toute sorte de choses – un nombre, une figure, un objet mathématique complexe, la réponse à une question concernant une propriété d'une classe d'objets, ou encore une preuve – notamment lorsqu'il s'agit, dit Hersh, de montrer qu'il n'y a pas de solution du problème demandé.

Ce n'est donc que dans des cas particuliers que la preuve devient l'objet même de la recherche ; dans les autres cas, elle est un objet second, qui est requis pour

13. Sur ce courant, voir plus haut dans ce chapitre, note 12.
14. (Hersh 1997, p. 6).
15. (ibid., p. 73).

obtenir la *validation* de l'assertion. Les preuves sont en effet d'abord des arguments jugés concluants par les mathématiciens :

> La découverte mathématique [= la résolution de problème] repose sur une validation appelée « preuve », l'analogue de l'expérience en science physique. Une preuve est un argument concluant qu'un résultat proposé découle d'une théorie acceptée. « Découle » signifie que l'argument convainc des mathématiciens qualifiés et sceptiques. Je donne ici une définition volontairement sociale de la « preuve ». Une telle définition n'est pas conventionnelle, mais elle est tout à fait conforme à la réalité.

Même en mettant de côté cette définition sociale de la preuve, le rapport que Hersh indique avec la notion de problème n'est pas la plus communément tenue par les mathématiciens – ce pour quoi il consacre sans doute un chapitre entier de son ouvrage à cette notion. Il est plutôt admis en général, conformément à la tradition antique de la géométrie euclidienne (sur laquelle nous revenons plus loin), qu'il y a deux grandes formes de problèmes, les *problèmes de trouver* et les *problèmes de prouver* (*problems-to-find* et *problems-to-prove*), que le mathématicien Pólya a bien distinguées :

> 1. Le but d'un « problème de trouver » est de trouver un certain objet, l'inconnue du problème. L'inconnue est aussi appelée « quaesitum », c'est-à-dire la chose recherchée, ou la chose requise. Les « problèmes de trouver » peuvent être théoriques ou pratiques, abstraits ou concrets, des problèmes sérieux ou de simples énigmes. Nous pouvons chercher toutes sortes d'inconnues ; nous pouvons essayer de trouver, d'obtenir, d'acquérir, de produire ou de construire toutes sortes d'objets imaginables. Dans le problème de l'histoire mystérieuse, l'inconnue est un meurtrier. Dans un problème d'échecs, l'inconnue est un coup des joueurs d'échecs. Dans certaines énigmes, l'inconnue est un mot. Dans certains problèmes élémentaires d'algèbre, l'inconnue est un nombre. Dans un problème de construction géométrique, l'inconnue est une figure.
>
> 2. Le but d'un « problème de prouver » est de montrer de façon concluante qu'une certaine affirmation clairement énoncée est vraie, ou bien de montrer qu'elle est fausse. Il s'agit de répondre à la question : Cette affirmation est-elle vraie ou fausse ? Et nous devons répondre de manière concluante, soit en prouvant que l'affirmation est vraie, soit en prouvant qu'elle est fausse. Un témoin affirme que l'accusé est resté chez lui une certaine nuit. Le juge doit déterminer si cette affirmation est vraie ou fausse et, en outre, il doit motiver le plus possible sa conclusion. Le juge a donc un « problème de prouver »[16].

16. (Pólya [1945] 1990, p. 154).

La différence entre Pólya et Hersh est subtile mais importante : pour Pólya, trouver le criminel n'est pas un travail de même nature que montrer que l'accusé n'est pas resté chez lui la nuit du crime. Pour Hersh au contraire, trouver le criminel et prouver qu'il l'est sont deux aspects d'un même acte, le second étant nécessaire à l'achèvement du premier. Se contenter du premier point, c'est simplement trouver un suspect ; prouver que l'accusé n'était pas chez lui la nuit du crime est une preuve intermédiaire dans la preuve globale qui convainc le criminel – qui le trouve vraiment.

Autrement dit, pour Hersh, toute investigation mathématique est un problème-de-trouver, qu'il s'agisse d'un nombre, d'une figure, ou de la vérité ou de la fausseté d'une proposition ; et toutes ces recherches doivent, par ailleurs, être justifiées par des preuves plus ou moins rigoureuses. La distinction que fait Pólya vaut surtout pour l'enseignement des mathématiques, comme le montre l'exemple du théorème de Pythagore qu'il utilise peu après : prouver ce théorème est un exercice scolaire, car sa vérité n'est pas vraiment en cause. La distinction de Pólya porte d'abord sur la forme des *exercices* canoniques qu'on présente à l'élève, mais dans le cadre de la recherche des mathématiciens professionnels, *qui ne savent pas* si une conjecture est vraie ou fausse, c'est l'articulation proposée par Hersh entre problème et preuve qui semble la plus pertinente.

§ 147. Les formes de problème

Afin de mieux analyser ces rapports entre preuve et problème, il est utile de considérer quelques exemples. Nous commençons par un problème-de-trouver simple, celui de trouver le double d'un nombre. On peut définir les entiers naturels de manière récursive en posant l'élément nul et en définissant tout autre entier comme le successeur d'un autre :

$$\mathbf{0} \in \mathbb{N}$$
$$\mathbf{succ}(n) \in \mathbb{N}\ [n \in \mathbb{N}]$$

On peut alors définir la fonction **double** de manière récursive comme suit :

$$\mathbf{double}(\mathbf{0}) = \mathbf{0}$$
$$\mathbf{double}(\mathbf{succ}(n)) = \mathbf{succ}(\mathbf{succ}(\mathbf{double}(n)))$$

Cette définition dit que, pour calculer le double d'un entier, il faut ajouter 2 (exprimé par **succ**(**succ**())) au double de l'entier précédent, et pour **0**, prendre **0**.

Considérons le problème de trouver x tel que $x = \mathbf{double}(5)$. Ici, la solution est très simple : il suffit d'appliquer les règles définissant **double**() :

$$\mathbf{double}(5)) = \mathbf{double}(4) + 2 = \mathbf{double}(3) + 2 + 2 = \text{etc.}$$

Ici, on peut dire que la proposition $P1(x)$ définie par $x = \mathbf{double}(5)$ est la *spécification* de x. Le calcul est ici la preuve que 10 répond à cette spécification. Si on avait donné la réponse intuitivement, le calcul aurait servi de *vérification*.

L'établissement de la vérité d'une proposition peut se faire de manière similaire. Par exemple, on peut définir la propriété de parité d'un entier $E(x) \equiv \mathbf{pair}(x)$ par les deux règles suivantes :

$$\text{ev0} : \mathbf{pair}(0)$$
$$\text{evn} : \mathbf{pair}(n) \rightarrow \mathbf{pair}(\mathbf{succ}(\mathbf{succ}(n)))$$

$E(10)$ peut être vu comme un problème similaire à $P1(5)$, dont on peut donner une réponse intuitive, et vérifier ensuite la vérité par l'application de la règle evn cinq fois et celle finale de ev0, ou qu'on peut calculer de manière similaire.

Voici à présent un second problème, légèrement plus compliqué que le précédent : trouver x tel que $18 = \mathbf{double}(x)$. Appelons $P2(x)$ cette spécification. Ici le calcul de la fonction **double**() ne nous sert à rien pour résoudre ce problème, il ne peut servir qu'à vérifier la solution. Celle-ci doit être trouvée par un autre moyen, soit intuitivement, soit par un autre calcul. Dans le cas où le problème est $17 = \mathbf{double}(x)$, la nature du problème change : il faut *montrer* qu'il n'y a pas de solution, par exemple en énumérant les doubles de tous les entiers jusqu'à ce qu'ils dépassent 17.

Voici un troisième problème : trouver, pour tout $y \in \mathbb{N}$, x tel que $y = \mathbf{double}(x)$. Ici la spécification, qu'on peut appeler $P3(y, x)$, est une fonction de y, et on doit donc trouver une solution générale, une fonction $g(y)$ telle que $y = \mathbf{double}(g(y))$. Dans ce type de problème il n'y a pas en général de méthode infaillible pour déduire la solution. Celle-ci doit être trouvée puis vérifiée. Par exemple la fonction partielle $\mathbf{demi}(y)$ définie de manière récursive, pour les entiers pairs uniquement, est candidate :

$$\mathbf{demi}(\mathbf{0}) = \mathbf{0}$$
$$\mathbf{demi}(\mathbf{succ}(\mathbf{succ}(n))) = \mathbf{succ}(\mathbf{demi}(n)))$$

Cette définition dit que pour calculer la moitié d'un nombre, il faut ajouter 1 à la moitié du nombre deux fois précédent (quand elle existe) et pour **0**, prendre **0**. Cependant il faut *prouver* que cette fonction est bien conforme à la spécification, car rien dans cette définition ne l'indique. En d'autre termes, il faut prouver :

$$\forall y, \mathbf{pair}(y) \rightarrow P3(y, \mathbf{demi}(y))$$

Nous ne donnons pas cette preuve par souci de concision.

Enfin, (quatrième forme de problème) on peut vouloir établir une propriété générale pour les entiers, comme celle-ci :

$$P4 : \forall x \in \mathbb{N}, \mathbf{pair}(\mathbf{double}(x))$$

Ici, il s'agit de savoir si le double de tout entier est pair : c'est la réciproque de la proposition précédente. On peut donc la voir d'une manière similaire, comme affirmant que toute solution du problème calculatoire **double**(n) pour n quelconque, est aussi une solution du problème **pair**(x). Sa preuve consiste aussi en une vérification universelle.

Ces quatre exemples ont des points communs. Pour les problèmes P1 à P3 il a fallu *trouver* la solution, même si dans P1, la méthode pour ce faire était donnée. Pour P2 et P3, la recherche était heuristique, et il a fallu de ce fait vérifier la solution *a posteriori*. Cette situation n'est pas agréable, et P3 a d'ailleurs pour visée la réduction de P2 à une recherche de type P1 (puisqu'il n'y a plus alors qu'à appliquer la fonction **demi**()). Le problème P4 requiert seulement une vérification. Seulement, comme pour P3, et contrairement à P2, cette vérification est paramétrique, c'est-à-dire qu'on doit trouver une méthode universelle pour faire cette vérification.

Ce dernier point suggère de voir également P4 comme un problème-de-trouver similaire aux trois premiers. Ce point de vue permettrait de rendre plausible la thèse de Hersh, que tout problème est un problème-de-trouver, et que la preuve est un processus par lequel on se convainc que la solution est correcte, ce que nous avons appelé vérification. Seulement, dans les problèmes P3 et P4, il faut trouver la méthode de vérification elle-même !

C'est là qu'intervient la théorie des types, qui tente de fonder logiquement cette possibilité. On le sait, les méthodes de vérification requises par P3 et P4 procèdent par récurrence. Nous la donnons rapidement pour P4 :

Démonstration. Il faut d'abord prouver que la propriété recherchée est vraie pour

0 : **pair**(**double**(**0**)). On a :

pair(**0**)	ev0
⇒**pair**(**double**(**0**))	Déf. **double**()

Il faut ensuite prouver que si la propriété est vraie pour n : **pair**(**double**(n)), alors elle est aussi vraie pour **succ**(n) : **pair**(**double**(**succ**(n))). On a :

pair(**double**(n))	Hyp. de récurrence
⇒**pair**(**succ**(**succ**(**double**(n))))	evn
⇒**pair**(**double**(**succ**(n)))	Déf.**double**()

□

La théorie des types propose de considérer que cette démonstration par récurrence est elle-même un objet qu'on a trouvé. Sa vérification est un calcul, similaire à celui qu'on avait fait dans le problème P1. Simplement, les règles de calcul ne sont pas ici celles de **double**(), mais celles de $\mathbb{N}$ lui-même, c'est-à-dire les règles du raisonnement par récurrence qui le caractérisent.

Une telle proposition mène à son terme l'idée hilbertienne qu'il est possible d'étudier mathématiquement les preuves mathématiques. Cette particularité, qui relève du projet fondationnel de la théorie des types, est bien exposée dans le texte suivant :

> Nous notons qu'un fondement de la théorie des ensembles comporte deux « couches » : le système déductif de la logique du premier ordre et, formulés à l'intérieur de ce système, les axiomes d'une théorie particulière, telle que ZFC[17]. Ainsi, la théorie des ensembles ne concerne pas seulement les ensembles, mais plutôt l'interaction entre les ensembles (les objets de la deuxième couche) et les propositions (les objets de la première couche). En revanche, la théorie des types est son propre système déductif : elle n'a pas besoin d'être formulée à l'intérieur d'une superstructure, telle que la logique du premier ordre. Au lieu des deux notions de base de la théorie des ensembles, à savoir les ensembles et les propositions, la théorie des types a une seule notion de base : les types. Les propositions (énoncés que l'on peut prouver, réfuter, supposer, nier, etc.) sont identifiées à des types particuliers […]. Ainsi, l'activité mathématique consistant à

17. La théorie des ensembles de Zermelo-Fraenkel.

prouver un théorème s'identifie à un cas particulier de l'activité mathématique consistant à construire un objet – en l'occurrence, un habitant d'un type qui représente une proposition[18].

De toute cette analyse découle l'idée qu'il y a trois situations possibles face à un problème, et quatre formes de jugements.

1. Il y a les problèmes pour lesquels il faut simplement vérifier qu'une solution donnée est correcte, c'est-à-dire se conforme aux règles du problème. Ce sont des problèmes calculatoires. Le problème P1 rentre dans cette classe, car calculer la solution selon une méthode donnée est similaire à une vérification. La forme de jugement correspondante en théorie des types est simplement $a = b$ où a et b désignent deux calculs aboutissant au même résultat, par exemple $\mathbf{succ}(\mathbf{succ}(\mathbf{succ}(1))) = \mathbf{double}(2)$.
2. Il y a les problèmes (comme P2 ou P3) pour lesquels il faut trouver une solution, qu'il s'agisse d'un objet précis ou d'un objet paramétrique, c'est-à-dire une méthode pour calculer un objet en fonction des paramètres du problème. Si A est le problème, la forme de jugement correspondante en théorie des types est $a \in A$, où a est un objet ou une méthode.
3. Il y a enfin les « méta-problèmes », où il s'agit de vérifier que deux problèmes sont équivalents, c'est-à-dire qu'ils ont les mêmes solutions, ou qu'ils sont simplement reliés entre eux, comme P4 : $\forall x, y, P3(x, y) \Rightarrow E(y)$ où $E(y)$ désigne $\mathbf{pair}(y)$. La forme de jugement correspondante en théorie des types est $A = B$, où A et B sont deux problèmes. Quand ces problèmes sont paramétriques, il faut trouver une méthode de vérification.
4. Cela implique qu'il doit y avoir une quatrième forme de jugement, qui définit les règles par lesquelles une vérification paramétrique peut procéder. Si les paramètres sont en nombre fini, la règle est simple : vérifier tous les cas possibles les uns après les autres. Cependant elle ne peut s'appliquer pour les ensembles infinis. On la remplace, dans le cas entiers naturels, par la règle de la démonstration par récurrence, comme nous venons de le voir. Chaque type de paramètres a ainsi ses propres règles par lesquelles il peut être utilisé pour définir de nouveaux problèmes, que ceux-ci portent sur des objets de même type ou de type différent. Définir ces règles suppose bien entendu qu'on ait également défini l'ensemble des paramètres en question, c'est-à-dire qu'on ait une méthode pour parcourir tous ses éléments (il s'agit

18. (Univalent Foundations Program 2013, p. 17).

de **0** et **succ**(n) pour les entiers naturels). Aussi cette quatrième forme de jugement est tout simplement A set, « A est un ensemble », et il consiste en la stipulation de toutes ces règles.

Nous avons ainsi présenté, dans un ordre différent de celui suivi habituellement, les quatre formes de jugement que reconnaît la théorie constructive des types telle que l'a élaborée Per Martin-Löf[19]. Nous sommes à présent en mesure d'examiner la manière par laquelle cette théorie permet de concevoir l'identité des règles du jugement et de la transformation du jugement en mathématiques.

§ 148. La théorie des types

Cette identité apparaît très bien dans le tableau suivant[20], qui affirme, dans la tradition intuitionniste, l'identité des problèmes et des propositions, ou encore des constructions et des preuves :

	A ensemble	$a \in A$
1	A est un ensemble	a est un élément de l'ensemble A
2	A est une proposition	a est une preuve (construction) de la proposition A
3	A est une intention (attente)	a est une méthode pour satisfaire (réaliser) l'intention (attente) A
4	A est un problème (tâche)	a est une méthode pour résoudre le problème (faire la tâche) A

Les deux colonnes exposent chacune une des formes de jugement identifiées au développement précédent, à savoir la quatrième (qui est en général présentée d'abord) et la seconde. Chaque ligne en donne une expression différente, que Martin-Löf pose comme équivalentes.

Dans la première colonne, on voit ainsi le rapprochement des notions d'ensemble, de proposition et de problème, cette dernière notion étant également représentée par les termes d'*intention, attente, tâche*. Dans la seconde, ce sont les notions d'élément, de preuve (ou construction) et de méthode pour *satisfaire, résoudre, réaliser, faire* qui sont rapprochées.

Martin-Löf note que « l'analogie » entre les lignes 1 et 2 fait référence à la cor-

19. (Martin-Löf 1984, p. 5-10).
20. (ibid., p. 5). Les numéros sont ajoutés par nous. Nous avons omis la dernière colonne, qui ne nous est pas utile.

respondance établie par Curry et Feys, et généralisée par Howard[21]. Cette liaison est bien apparente dans nos deux premiers exemples : la propriété **pair**(x) peut être vue comme le prédicat d'une proposition, par exemple **pair**(6), ou comme l'ensemble de tous les x la satisfaisant. Pour les ensembles primitifs la proposition correspondante doit permettre de définir ce que signifie être membre de cet ensemble. Par exemple, pour les entiers naturels, on peut voir $\mathbb{N}$ comme la succession ordonnée d'éléments à partir d'un élément initial **0**.

Les concepts introduits aux lignes 3 et 4 du tableau correspondent selon Martin-Löf à la même idée fondamentale, introduite de manière concurrente par Heyting (première formulation) en 1931 et par Kolmogorov (deuxième formulation) en 1932 pour interpréter la logique intuitionniste[22]. Martin-Löf rapproche également l'interprétation de Kolmogorov de la programmation :

> « *a* est une méthode ... » peut être lu comme « *a* est un programme ... ». Puisque les langages de programmation ont une notation formelle pour le programme *a*, mais pas pour *A*, nous complétons la phrase avec « ... qui répond à la spécification *A* ». Dans l'interprétation de Kolmogorov, le mot problème fait référence à quelque chose à faire et le mot programme à comment le faire[23].

Il reste à établir le lien entre 1-2 et 3-4, entre proposition et problème. On voit bien d'une part pourquoi une proposition peut être assimilée à un problème, parce que tout problème doit décrire l'objet recherché, et cette description est une proposition, ou encore une vérification qu'on pourra faire de tout objet présenté, conduisant à un jugement de conformité. Mais d'autre part, une proposition générale, énonçant une propriété d'une classe d'objets, peut elle-même être prise pour problème, car il s'agit alors de montrer, comme nous l'avons vu, le lien entre deux spécifications, pour un entier par exemple l'une d'être pair, l'autre d'être le double d'un autre entier.

Ainsi le rapport entre le jugement et la transformation du jugement n'est rien d'autre que celui entre problème et preuve. Chacune des trois formes de jugement qui résout un problème requiert une preuve, qui est un acte de connaissance – qu'il s'agisse d'un calcul, de la construction d'une nouvelle méthode de calcul ou d'un raisonnement comparant des méthodes. Dans le premier cas, on applique

21. (ibid., p. 6). Sur les liens entre Howard et Martin-Löf, voir (Wadler 2015, p. 10). Sur la correspondance de Curry-Howard, voire (Bourdeau et Mosconi 2022, p. 680).
22. (Heyting 1931 ; Kolmogorov 1932).
23. (Martin-Löf 1984, p. 6).

des règles (le jugement *déterminant* de Kant[24]), dans le second on les élabore (le jugement *réfléchissant*), et dans le troisième on les organise entre elles (l'activité de la raison).

Prouver et construire sont d'abord des actes. Il ne faut les confondre ni avec leur résultat (le théorème, l'objet construit), ni avec la description de ces actes, qui peut indiquer comment les reproduire, mais nullement se substituer à eux[25]. Martin-Löf écrit :

> Le mot acte, bien sûr, est le mot le plus général que nous utilisons pour tous nos actes, et, de même, le mot objet a aussi cette généralité maximale. Ainsi nous avons des actes de conjecture, de doute, de souhait et de crainte et ainsi de suite, et nous avons les objets de ces actes, c'est-à-dire des objets de conjecture, de doute, de souhait, de crainte et ainsi de suite. Un type particulier d'actes et, corrélativement, un type d'objets vers lesquels nous nous dirigeons dans ces actes, sont respectivement des actes de connaissance et des objets de connaissance. Ma réponse aux questions, *Qu'est-ce qu'un jugement ?* et, *Qu'est-ce qu'une preuve d'un jugement ?* est simplement qu'une preuve d'un jugement est un acte de connaissance et que le jugement qu'elle prouve est l'objet de cet acte de connaissance, c'est-à-dire un objet de connaissance.

La théorie des types permet donc de montrer que l'acte et l'objet du jugement *requièrent* d'avoir les mêmes règles, car l'acte n'a de sens que relativement à son objet ; mais réciproquement cet objet sans l'acte qui le remplit est vide et entièrement problématique, comme l'exprime le texte suivant, qui compare les deux formes de jugement présentées dans le tableau ci-dessus :

> Connaître un jugement est la même chose qu'en avoir une preuve, et connaître un jugement de la forme particulière [A est vrai] est la même chose que savoir comment, ou être capable, de vérifier la proposition A […] tandis que la connaissance d'un jugement de la forme [A est une proposition] est la connaissance d'un problème, d'une attente ou d'une intention, qui est simplement *le savoir de ce qu'*il faut faire. […] Ainsi, la différence entre ces deux types de connaissances est la différence entre savoir quoi faire et savoir comment le faire. Et, bien sûr, il ne peut être question de savoir comment faire quelque chose avant de savoir ce qu'il faut faire. [26].

Dans les exemples du développement précédent, le savoir-quoi désigne la propriété $P1(x) : x = \mathbf{double}(5)$ ou $P2(x) : 18 = \mathbf{double}(x)$, le problème paramétrique $P3(y, x)$ ou encore l'assertion $P4$. Le savoir-comment désigne la vérifica-

24. Voir plus haut, § 135, et notre note 17 à ce sujet.
25. Voir par exemple (Troelstra 1969, p. 3), (Sundholm 1997).
26. (Martin-Löf [1983] 1996, p. 26).

tion que 10 est une solution dans le premier cas, 9 dans le second, et la fonction **demi**(y) dans le troisième. Dans le dernier cas enfin, il s'agit de vérifier le détail de la connexion qui existe entre les notions de parité et de doublement d'un entier.

§ 149. La lecture relationnelle des règles

On peut à présent comprendre pourquoi il n'est pas pertinent, dans cette perspective, de distinguer les jugements « théoriques », qui porteraient sur des faits ou sur des relations, des jugements « pratiques » qui diraient comment faire pour obtenir un résultat – un fait ou une relation. Le cas des entiers naturels – un cas certes minimal et pur, mais suffisant – montre pourquoi, au fond, ces deux jugements apparemment distincts sont identiques. Juger que 10 est le double de 5 porte certes sur une relation, mais l'établir exige un calcul ; on peut ne pas en ressentir le besoin parce que l'exemple est évident, mais cela devient patent lorsqu'on demande si $323022699 \div 7$ est un entier naturel : pour la plupart des personnes, juger d'une telle « relation » exigera de poser le calcul. De la même manière, la preuve qu'une méthode est correcte ou que deux méthodes sont équivalentes exige également de poser des calculs. Ceux-ci sont légèrement différents parce qu'ils ne se situent pas sur le même niveau, cependant ils s'appuient tous sur le même schéma opératoire de la récurrence, caractéristique des entiers naturels. C'est par ce schéma qu'ils sont eux-mêmes définis, ainsi que les propriétés et les fonctions qui les utilisent. Aussi, prouver quoi que ce soit à leur propos revient toujours à réduire les termes de la proposition en jeu à ce schéma, et à montrer qu'*il ne s'y trouvait rien d'autre qu'on n'y avait soi-même posé*.

Une note terminologique est ici nécessaire. *Induction* est le terme utilisé en mathématiques pour désigner en général ce que nous avons appelé *schéma opératoire*, et dont le principe de récurrence sur les entiers naturels est un exemple. L'induction est l'opération par laquelle on peut générer ou parcourir un ensemble d'objets. Elle est en théorie des types partie intégrante de la définition de cet ensemble, et elle est constituée d'un nombre fini de règles. L'induction peut être dépendante d'un autre type, lorsqu'un ensemble est construit à partir d'un autre, comme par exemple celui des listes de longueur n[27]. *Récursivité* et *induction* sont parfois utilisés de manière interchangeable. Nous prenons le parti de réserver le premier terme au principe d'induction qui vaut sur les entiers naturels. Quant à *générativité*, nous

27. Voir plus loin § 161.

utilisons ce terme pour caractériser les espaces ou structures qui, construits inductivement, sont par là infinis : ainsi les entiers, les arbres, etc. Il est à noter que nous ne prenons pas ce terme dans le sens strict qu'il a en théorie de la calculabilité, où il dénote une forme intermédiaire d'infinité procédurale, celle des grammaires de Chomsky.

Si cette interprétation est correcte, d'où vient alors qu'en mathématiques, et partant également dans de nombreuses théories scientifiques, l'aspect dynamique des règles soit occulté, de sorte qu'elles apparaissent comme régissant seulement des relations entre objets ?

Pour comprendre cela, il faut revenir à la notion de calcul comme forme élémentaire de la résolution d'un problème. Dans la situation la plus simple, on l'a vu, l'exécution d'une procédure connue « calcule » un résultat en fonction des paramètres du problème. Il est donc possible de voir toute fonction comme une simple relation extensive entre un ensemble de départ et un ensemble d'arrivée (où *départ* et *arrivée*, ou encore les flèches par lesquelles on représente souvent les fonctions, trahissent l'origine dynamique de la fonction). C'est ainsi d'ailleurs que les fonctions sont définies en théorie des ensembles. Il y a ainsi une ambiguïté essentielle de la notion de *fonction* en mathématiques[28] qui est passible d'une double interprétation, dynamique et relationnelle. Prenons par exemple une expression fonctionnelle simple, inspirée du λ-calcul*[29] :

Code 9.1 – Illustration des principes du λ-calcul

```
1    first of (sort (append('dog', 'rabbit') (sort ('mouse', 'cat'))))
2    → first of (sort (append('dog', 'rabbit') ('cat', 'mouse')))
3    → first of (sort ('dog', 'rabbit', 'cat', 'mouse'))
4    → first of ('cat', 'dog', 'mouse', 'rabbit')
5    → 'cat'.
```

Dans cette expression, `first of` désigne l'opération de sélectionner le premier terme d'une liste, `append`, celle de concaténer deux listes entre elles, et `sort` de trier les termes d'une liste par ordre alphabétique. Chacun opère sur la liste indiquée par les parenthèses qui le suivent.

Cette expression peut être interprétée de deux manières. L'une la voit comme une suite de constats, c'est-à-dire de jugements d'égalité entre deux termes successifs : la flèche $\rightarrow$ prend ici le sens de la copule *est*. Il s'agit, dans la terminologie

28. Voir à ce sujet la réflexion de (Granström 2011).
29. (Barendregt et Barendsen [1984] 2000).

de Kant, de jugements prédicatifs. L'autre la voit comme une suite de transformations dynamiques dues à l'exécution des instructions que représenteraient les opérateurs : la flèche $\rightarrow$ représente ici le déroulement d'un processus.

Le processus dont il est question ici n'est pas pour autant celui de phénomènes matériels, observés dans le monde. Il n'est rien d'autre que le progrès du jugement lui-même qui avance, ligne après ligne, jusqu'à la solution finale, comme les *prosyllogismes* de Kant[30]. Elle invite à voir le jugement lui-même comme un acte qui modifie le savoir du sujet.

Ce qui rend la lecture relationnelle de cette proposition également possible, c'est que cet acte n'est pas arbitraire : il obéit à une règle, qui définit ce qu'est la fonction. En tant qu'on observe la règle qui régit cet acte, on la voit comme une relation à laquelle il faut se conformer. Le point de vue relationnel prévaut en mathématiques contemporaines car elles s'occupent, à bien des égards, d'explorer les cadres possibles de la pensée systématique, et donc aussi essentiellement de l'étude des règles dans leur organisation systématique, et non de leur usage.

La forme relationnelle est privilégiée car elle permet de simplement consigner les résultats d'investigations mathématiques – entre points d'arrivée et points de départ, entre méthodes d'un même domaine, voire entre groupes de méthodes appartenant à des domaines distincts. Ce faisant ce qu'on perd, dans la forme relationnelle, c'est le temps du calcul lui-même, c'est-à-dire de l'exécution concrète des méthodes, qui n'intéresse pas les mathématiciens, et le temps de la recherche et de la délibération.

9.2 Interprétations de la géométrie

Cette justification de l'identité des règles du jugement et de la transformation du jugement, qui les voit comme deux moments d'un même *acte* convient-elle à l'investigation mathématique en général ? Dans cette section, nous la mettons à l'épreuve d'une théorie particulière, la géométrie euclidienne. Les raisons de ce choix sont multiples. D'abord, la géométrie nous est apparue historiquement déterminante dans l'avènement de la forme de rationalité que nous cherchons à circonscrire. D'autre part, sous sa forme classique, ses propositions se présentent sous une double forme, celle des problèmes qui demandent de construire des figures, et celle des théorèmes qui énoncent des vérités à leur propos. Cette dualité, qui fut dès

30. Voir plus haut, p. 524.

l'origine objet de discussion, est un défi pour notre solution, car elle accorde crédit à la dichotomie proposée par Pólya, entre problèmes de trouver et problèmes de prouver. Enfin, comme nous l'avons déjà souligné, la géométrie euclidienne est l'objet de recherches constructives depuis une trentaine d'années, sans doute du fait qu'elle est souvent invoquée comme figure tutélaire de l'ambition d'une refondation constructive des mathématiques[31]. Ces recherches, qui furent débutées par Mäenpää et Von Plato (1990) et poursuivies par Beeson (2010), Naibo (2015), Naibo et Panza (2021) et Von Plato (1995) entre autres, sont toujours actives, ce qui montre que la géométrie euclidienne « résiste », en quelque sorte, à ces formalismes. Il nous intéresse de comprendre pourquoi.

Nous rappelons dans un premier développement (§ 150) la controverse rapportée par Proclus concernant la dualité des problèmes et des théorèmes, puis son interprétation célèbre par l'historien des mathématiques Wilbur Knorr qui tranche en faveur de la primauté des problèmes en invoquant l'arrière-plan pratique sur fond duquel se serait développée la géométrie grecque (§ 151). Nous exposons enfin une des tentatives de réexpression constructive de la géométrie due à Mäenpää et Von Plato (§ 164), et détaillons l'un des exemples qu'ils proposent, concernant la construction d'un parallélogramme (§ 153). Le dernier développement de cette section (§ 154) est consacré à l'examen de ses faiblesses, ce qui nous amènera, à la section suivante, à revoir de manière assez importante la réponse qui vient d'être dégagée concernant la dualité des règles du jugement et de l'action.

§ 150. Théorèmes et problèmes dans les *Éléments*

En première approche, un théorème énonce une propriété générale des objets géométriques considérés, qu'il faut démontrer, tandis qu'un problème demande de construire, à partir d'objets initiaux donnés, un objet en relation avec eux comme par exemple, construire le triangle circonscrit au cercle $\mathcal{C}$[32]. Pappus définit ainsi les problèmes et les théorèmes à partir de leurs racines verbales :

> Il convient d'appeler problème ce dans quoi l'on propose (*proballetai*) de réaliser ou de construire quelque chose, et théorème ce dans quoi, une fois certaines suppositions faites, on en perçoit (*theoreitai*) la conséquence et, d'une manière générale, ce qui les affecte[33].

31. Voir plus haut, § 145.
32. (Knorr 1983, p. 127), (Sidoli 2018, p. 405)
33. Pappus, *Collection*, livre III, introduction, trad Ver Ecke, p. 21.

Le problème serait donc un *projet*, et le théorème une *contemplation* : cette distinction, qui semble transposer l'opposition théorie / pratique au sein même des mathématiques, a encore le statut d'évidence aux Temps modernes, comme l'exprime laconiquement cette définition du *Dictionnaire* de Furetière :

> Théorème, subst. masc. Vérité démontrée et déterminée, ou proposition qui s'arrête à la spéculation, et dans laquelle on considère les propriétés des choses toutes faites. Par opposition à problème, qui y ajoute la pratique, et la construction. La Géométrie se distribue en théorèmes, et problèmes[34].

La controverse antique provient de ce que, tandis que les théorèmes semblent bien rendre compte d'une « expérience théorique », les problèmes rapprochent la géométrie des arts productifs, ce qui est inacceptable pour les partisans des Idées. L'école de Speusippe rejette ainsi la résolution de problèmes comme une formulation impropre de leur contenu mathématique, en faveur des théorèmes. En effet,

> [Les Mathématiques] traitent des choses éternelles, parmi lesquelles il n'y a pas de génération. Ainsi un problème n'a pas place auprès d'elles, puisqu'il propose d'engendrer ou de produire quelque chose qui n'existait pas auparavant[35].

Cette position prend son origine chez Platon lui-même, qui s'offusquait de l'usage de verbes d'action en mathématiques :

> — Ceux qui ont ne serait-ce qu'une petite expérience de la géométrie ne nous le disputeront pas : c'est que cette connaissance est tout à l'opposé de ce qu'en disent les discours tenus par ceux qui la pratiquent.
> — Comment cela ? dit-il.
> — C'est qu'ils en parlent de façon bien risible, et bien utilitaire en effet, c'est comme des gens de pratique, et en vue d'une pratique, qu'ils produisent tous leurs énoncés, parlant de « porter au carré », d'« appliquer » et d'« additionner », et énonçant tout sur ce mode ; alors que tout cet enseignement, on ne s'y exerce en fait que pour parvenir à la connaissance […] de ce qui est toujours, et non de ce qui à un certain moment naît et se défait, […][et] qu'il est propre à tirer l'âme vers la vérité, et à façonner la réflexion du philosophe pour lui faire orienter vers le haut ce qu'à présent nous orientons indûment vers le bas[36].

Platon suggère ici que cette forme malencontreuse d'expression rend compte des exercices préparatoires à la connaissance, qui doivent s'achever dans une contem-

34. Antoine Furetière, *Dictionnaire universel*, t. 3.
35. Proclus, *Commentaire du premier livre des* Éléments *d'Euclide*, traduction de G.R. Morrow, p. 77-78.
36. Platon, *République*, 527a-b, traduction de Pierre Pachet.

plation impassible. L'entendement humain doit s'appuyer sur l'imagination pour se représenter les figures géométriques comme si elle étaient produites dans le temps[37], mais l'intellection pure doit s'en passer. Les problèmes seraient donc les expressions pédagogiques d'un contenu objectif qui pourrait être reformulé en théorèmes. Proclus attribue à l'école de Zénodote[38] l'idée selon laquelle les constructions visent à démontrer l'existence des objets dont les propriétés sont étudiés par les théorèmes. Les problèmes seraient ainsi des « lemmes d'existence », préliminaires et subordonnés aux théorèmes.

D'une certaine manière, l'algébrisation de la géométrie à partir du XVII^e^ siècle a sanctionné cette opinion, et les problèmes de construction ont été largement interprétés, au siècle dernier, comme des théorèmes d'existence, notamment suite à l'article influent de ZEUTHEN (1896). Ainsi les axiomatisations de la géométrie euclidienne par Hilbert et Tarski sont constituées uniquement de théorèmes. Dans la théorie de Tarski, les seuls objets donnés sont les points, ainsi que leurs relations d'intermédiarité (*betweenness*) entre trois points et de congruence entre deux paires de points[39]. Les objets géométriques complexes (lignes, triangles, etc.) qui seraient des ensembles définis à l'aide de ces relations fondamentales, ne sont pas nécessaires à l'expression du contenu de la géométrie car, comme le remarque Tarksi, ils sont toujours définis à partir d'un nombre fini de points, et peuvent donc être directement exprimés par leurs relations.

Une telle représentation de la géométrie revient à faire de toutes ses propositions des théorèmes, puisque toute demande de construction d'un objet complexe y est représentée comme celle d'une relation entre points. Par exemple, la proposition I.1 d'Euclide, qui demande de construire un triangle équilatéral, s'exprime dans la géométrie de Tarski comme suit :

$$\forall a, b, \exists c : \neg Col(a, b, c) \wedge ab \equiv ac \equiv bc$$

Dans cette proposition, a, b, c sont des points, et $\neg Col(a, b, c)$ signifie qu'ils sont distincts et non alignés (la relation de colinéarité $Col(a, b, c)$ est définie à partir de la relation élémentaire d'intermédiarité (*betweenness*)[40]. Enfin, $\equiv$ designe la relation élémentaire de congruence entre segments.

37. Proclus, ibid., p. 78.
38. ibid.,p. 80.
39. (TARSKI 1959, p. 17).
40. (ibid., p. 17)), (BEESON 2015, p. 1205)

Ces axiomatisations s'inscrivent dans un mouvement plus large s'amplifiant à partir du XIX^e^ siècle où une conception purement théorique des mathématiques semble prendre l'avantage, avec notamment la mise en avant de la distinction « mathématiques pures / appliquées », ce qui a pour implication de rejeter hors des mathématiques « proprement dites » ses dimensions pratiques, orientées vers la solution de problèmes concrets, notamment technologiques[41].

§ 151. L'interprétation pragmatique de Knorr

Cependant cette réaction semble oublier une autre conception possible des théories axiomatiques, qui les fonderait sur les problèmes plutôt que sur les théorèmes. Proclus évoque en effet une opinion opposée à celle de Speusippe concernant la prééminence des théorèmes et des problèmes :

> D'autres au contraire, comme les mathématiciens de l'école de Ménèchme, trouvaient juste de dire que tout est problème, mais qu'il y en a deux sortes : tantôt il s'agit de fournir l'objet recherché, tantôt de voir, concernant un objet déterminé, ce qu'il est, de quelle espèce il est, quelles qualités il possède, ou encore quelles relations il entretient avec autre chose[42].

Proclus ne donne malheureusement pas plus de détail. Knorr[43] interprète la position de Ménèchme comme représentative d'une primauté double des problèmes sur les théorèmes pour les anciens géomètres. Les méthodes de construction de figures ont d'abord une primauté épistémologique, car elles ne seraient pas seulement l'objet des problèmes, mais serviraient également de manière centrale à la démonstration des théorèmes. Mais elles ont également une primauté d'intérêt. Pour Knorr, leur découverte est centrale à l'activité géométrique, et « la compilation de théorèmes lui était subordonnée[44] ». L'objectif principal des mathématiciens, selon lui, est de montrer la réduction possible d'un problème donné de construction à des problèmes déjà résolus, jusqu'à parvenir ainsi, par réductions successives, aux postulats décrivant les constructions élémentaires[45]. Sidoli (2018) développe cette idée en montrant la constructivité strictement modulaire des problèmes dans les livres I à VI des *Éléments*, en s'inspirant d'ailleurs de la modularité des programmes informatiques.

41. Hacking (2014) fait la généalogie et la critique de cette conception.
42. Proclus, ibid.,p. 78.
43. (Knorr 1983, p. 351).
44. (Knorr 1986, p. 360).
45. (Knorr 1983, p. 129).

Knorr précise que cet intérêt heuristique était suscité par la résolution pratique de questions empiriques, comme celles concernant le calcul des trajectoires célestes sur des modèles sphériques, ou des équilibres statiques entre masses[46]. L'abstraction de ces questions dans la formulation pure des problèmes géométriques n'aurait pas été due à une quelconque hypothèse philosophique quant à la nature mathématique du réel, mais à la constitution progressive du champ mathématique comme discipline autonome. Cet intérêt heuristique fondamental explique que la plupart des mathématiciens, y compris Archimède, étaient ouverts aux méthodes « mécaniques » de construction de figures, contrevenant sans états d'âme l'interdiction de Platon d'utiliser autre chose que la règle et le compas en géométrie[47].

La position inverse, qui affirme la prééminence des théorèmes, aurait été importée dans le champ mathématique par Platon, soucieux d'y faire l'illustration de sa métaphysique :

> Aveuglés par l'éclat de l'ontologie de Platon, les commentateurs tardifs cherchent à imposer à la géométrie classique une interprétation qu'elle ne soutient pas aisément. Puisque les problèmes cherchent à construire des entités géométriques, c'est-à-dire amener à l'être des choses qui existent éternellement, [...] ils doivent prendre une position subordonnée aux théorèmes. Seuls ces derniers ont en effet une forme philosophiquement correcte, prouvant ce qui est vrai des entités idéales de la géométrie. On concède aux problèmes un rôle à jouer dans l'état provisoire qu'est la connaissance incomplète, ou pour la commodité des élèves. Ou encore, ils peuvent rendre le service d'effectuer des preuves de l'existence de termes nécessaires dans les démonstrations formelles des théorèmes[48].

Ce dernier point en particulier, repris comme nous l'avons vu par Zeuthen (1896), est farouchement remis en cause par Knorr. Il montre d'abord que les problèmes proposent la plupart du temps des constructions d'objets dont l'existence n'est absolument pas en question, comme construire le milieu d'un segment[49]. Réciproquement, lorsque l'existence d'un objet est douteuse, les géomètres développent un *diorisme*, une forme tenue pour intermédiaire entre théorème et problème, qui vise justement à dégager le conditions de possibilité de la construction. Enfin, on trouve chez Euclide des théorèmes qui traitent de questions d'existence[50]. Selon Knorr, Proclus aurait projeté sur Euclide le schéma de questionnement aristoté-

46. (Knorr 1989).
47. (Knorr 1983, p. 22).
48. (Knorr 1986, p. 360).
49. (Knorr 1983, p. 129, 132).
50. (ibid., p. 131).

licien (s'enquérir de l'existence d'une chose avant de le faire de ses propriétés), de la même manière que Zeuthen aurait projeté sur la tradition géométrique le questionnement mathématique qui lui était contemporain, concernant l'assertion d'existence des objets mathématiques.

L'interprétation de Knorr a suscité de nombreuses recherches historiques, et été nuancée par Sidoli et Saito (2009). Si l'attention portée à la faisabilité pratique des constructions caractérise celles qui sont utilisées dans la résolution de problèmes, il n'en est pas de même pour celles utilisées dans la démonstration de théorèmes. En d'autres termes, si la fonction des problèmes était bien de donner des indications concernant le tracé effectif des figures avec les outils disponibles, la notion de construction est plus large, et peut aussi s'entendre comme une manière de *faire voir* les propriétés cachées dans l'exposition initiale d'une figure. Ainsi, pour Sidoli (2018, p. 405) qui fait le point sur cette question « il semble qu'il y ait encore, malgré tout, en raison du poids accordé aux théorèmes, une tendance générale lors de la lecture des *Éléments*, à voir les problèmes comme auxiliaires aux théorèmes. »

§ 152. La proposition de Mäenpää et Von Plato

Il existe une seconde interprétation, constructive cette fois, de la primauté des problèmes sur les théorèmes. Mäenpää (1997), un élève de Martin-Löf, dénonce ainsi l'interprétation axiomatique de la géométrie qui la fonde sur la logique des prédicats :

> La notion d'axiomatisation abstraite de Hilbert, telle qu'elle est illustrée dans ses *Grundlagen der Geometrie* (1899), réduit la géométrie à la preuve de théorèmes, en réduisant l'existence d'objets mathématiques à la cohérence du système axiomatique qui les définit implicitement. […] Le modèle de Hilbert s'est généralisé à travers les mathématiques de ce siècle, et les a réduites à la preuve de théorèmes. la résolution de problèmes, qui était la préoccupation première des mathématiciens grecs (Knorr 1986, ch. 8), a été écartée[51].

L'entreprise constructive, au contraire, vise selon lui à restituer le contenu même de la géométrie euclidienne, c'est-à-dire ses constructions et ses démonstrations en suivant fidèlement leurs arguments, plutôt qu'à refonder ses résultats par d'autres moyens axiomatiques et inférentiels. Nous allons dans ce développement suivre ce projet qu'il développe avec son collègue Von Plato dans un article de 1990, et

51. (Mäenpää 1997, p. 210).

que ce dernier reprend en 1995[52]. Les formalisations bien plus précises de Beeson se placent d'emblée dans un cadre hilbertien (2010) ou tarskien (2015), ce qui les rend moins pertinentes pour analyser la distinction entre théorème et problème.

L'idée de Mäenpää et Von Plato est la suivante :

> Les problèmes, les propositions et les ensembles sont identifiés conceptuellement dans la théorie des types, ils sont des types égaux. [...] En conséquence, les éléments d'un ensemble, les preuves d'une proposition et les solutions d'un problème sont identifiés en théorie des types (Martin-Lof 1984, pp. 5-13). La résolution de problèmes géométriques implique une phase de démonstration (apodeixis) en plus d'une phase de construction. On construit d'abord un objet et on démontre ensuite qu'il est bien du type requis. Par conséquent, une logique des problèmes doit permettre de prouver des propositions et d'effectuer des constructions. La théorie intuitionniste des types les traite comme des activités formellement identiques. Elle exprime également les constructions de manière explicite, ce qui permet de refléter formellement la division des preuves géométriques en phases de construction et de démonstration[53].

Le premier et le troisième postulats s'écrivent par exemple avec les règles d'introduction suivantes :

$$\text{Post.1} : \frac{a : \text{Point} \quad b : \text{Point}}{\text{ln}(a, b) : \text{Line}} \qquad \text{Post.3} : \frac{a : \text{Point} \quad r : \text{Line}}{\text{c}(a, r) : \text{Circle}}$$

La première règle d'introduction dit qu'à partir de deux objets Point a et b on peut introduire un objet Line $\text{ln}(a, b)$, et la seconde dit qu'à partir d'un Point a et d'une ligne r on peut construire un objet Circle $\text{c}(a, r)$. Tous ces objets sont associés à des actes bien précis qui les engendrent : *marquer* un point, *tracer avec la règle* une droite, *tracer avec le compas* un cercle. Ce sont ces actes que désignent les règles d'introduction. La seconde veut dire, par exemple : « *Si vous disposez d'un point et d'une ligne, alors on suppose que vous savez comment tracer au compas le cercle ayant pour centre ce point et pour rayon cette ligne* ».

L'ambition de Mäenpää et Von Plato est de montrer, en suivant Euclide, que tous les autres actes de construction dérivent de ceux-là, comme par exemple, tracer un triangle équilatéral ou un parallélogramme.

Von Plato modélise les propriétés des figures, comme celle de distinction entre deux points $\text{DiPt}(a, b)$ ou entre deux lignes $\text{DiLn}(l, n)$, de séparation d'une ligne

52. (Mäenpää et Von Plato 1990 ; Von Plato 1995).
53. (Mäenpää et Von Plato 1990, p. 280).

et d'un point Apt(a, l) (*apartness*), qui signifie qu'un point n'est pas situé sur une ligne, ou celle de convergence entre deux lignes $Con(l, m)$. Il définit une vingtaine d'axiomes, comme par exemple[54] :

$$\begin{aligned}
&\vdash \text{DiPt}(a, b) && \rightarrow \vdash \text{DiPt}(a, c) \vee \text{DiPt}(b, c);\\
&\vdash \text{DiLn}(l, m) && \rightarrow \vdash \text{DiLn}(l, n) \vee \text{DiLn}(m, n);\\
&\vdash \text{Con}(l, m) && \rightarrow \vdash \text{Con}(l, n) \vee \text{Con}(m, n).
\end{aligned}$$

Il s'agit là d'axiomes de séparation, qui disent que si deux points sont distincts, alors un troisième point est distinct au moins de l'un d'entre eux, qu'il en est de même avec les lignes, et enfin que si deux lignes sont convergentes, alors une troisième est convergente avec au moins l'une d'entre elles. Il est important de noter que ces relations sont des types, au même titre que les objets Point, Line. Elles peuvent entrer en tant que conditions dans la construction de nouveaux objets, comme ici, pour marquer un point d'intersection entre deux lignes convergentes :

$$\frac{l : \text{Line} \quad m : \text{Line} \quad \vdash \text{Con}(l, m)}{\text{pt}(l, m) : \text{Point}}$$

Il est à noter que dans sa reprise de 1995, Von Plato réécrit le postulat 1 d'Euclide en y ajoutant la condition que a et b soient distincts :

$$\text{Post.1} : \frac{a : \text{Point} \quad b : \text{Point} \quad \vdash \text{Dipt}(a, b)}{\text{ln}(a, b) : \text{Line}}$$

Nous y reviendrons par la suite.

Ce choix permet à Mäenpää et Von Plato de proposer l'unification entre théorème et problème dans une seule forme de jugement, qu'ils appellent *problème*[55] :

$\exists(y : B(x)) : C(x, y)$	:	problem $(x : A)$
	où :	
$x : A$	est	le *donné*,
$y : B(x)$	est	*ce qui est cherché*,
$C : (x, y)$	est	la *condition* (ou *spécification*).

Les théorèmes, dit Mäenpää sont donc des cas particuliers de problèmes où l'élément $y : B(x)$ est nul, si bien qu'on peut écrire le problème simplement :

$C(x)$: theorem $(x : A)$

54. (Von Plato 1995, p. 172).
55. (Mäenpää 1993, p. 90).

Afin de bien comprendre ce que signifie cette formalisation, nous allons suivre un exemple de problème que détaille Von Plato, celui de la construction d'un parallélogramme[56].

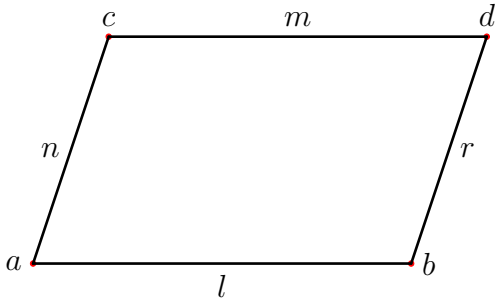
§ 153. La construction du parallélogramme

Les données d'entrée du problème du parallélogramme (le x de Mäenpää) sont trois points non alignés, qui s'écrivent ainsi dans le formalisme de Von Plato :

$$(a, b, c : \text{Point}; \tag{1}$$
$$\vdash \text{DiPt}(a, b); \tag{2}$$
$$\vdash \text{Apt}(c, \ln(a, b))). \tag{3}$$

– où la première assertion énonce que a, b, c sont des points, DiPt(a, b) (« distinct points ») la propriété que a et b sont deux points distincts et Apt$(c, \ln(a, b))$ (« apart ») celle que le point c n'est pas situé sur la droite $\ln(a, b)$. En d'autres termes, ces trois formules invitent à considérer trois points a, b, c distincts non alignés. Von Plato appelle ceci le *contexte initial* du problème. L'objet recherché

FIGURE 9.1 – Construction d'un parallélogramme



$y : B(x)$ est un parallélogramme, qui peut être défini comme une construction de quatre points et quatre droites :

$$a, b, c, d : \text{Point}; \tag{1}$$
$$l, m, n, r : \textbf{Line}; \tag{2}$$

56. (VON PLATO 1995, p. 186-188).

L'objet $B(x)$ doit répondre aux conditions $C(x)$ qui sont les suivantes :

$$\vdash \text{DiPt}(a, b); \vdash \text{DiPt}(c, d); \vdash \text{DiPt}(a.b, c.d); \tag{3}$$

$$\vdash \text{Inc}(a.b, l); \vdash \text{Inc}(c.d, m); \tag{4}$$

$$\vdash \text{Inc}(a.c, n), \vdash \text{Inc}(b.d, r); \tag{5}$$

$$\vdash \text{Par}(l, m); \vdash \text{Par}(n, r); \tag{6}$$

$$\vdash \text{DiLn}(l, m); \vdash \text{DiLn}(n, r). \tag{7}$$

(3) exige que les points sont distincts deux à deux, (4) et (5) que les quatre droites sont bien celles qui relient les quatre points entre eux (Inc = « incidentes »), et enfin (6) et (7) que les droites sont parallèles et distinctes deux à deux[57].

La résolution du problème procède comme suit :

1. À partir de a, b, c les droites $\text{ln}(a, b)$ et $\text{ln}(a, c)$ peuvent être construites par le postulat 1 mentionné plus haut.
2. À partir du point c et de la droite $\text{ln}(a, b)$, une droite parallèle à cette dernière, passant par c, peut être construite, et une autre à partir du point b et de la droite $\text{ln}(a, c)$. Cette procédure est définie par la règle de construction suivante :

   $$\frac{l : \text{Line} \quad a : \text{Point}}{\text{par}(l, a) : \text{Line}}$$

   Von Plato en fait un postulat dans son axiomatique, pour des raisons fondationnelles, mais cette procédure est bien construite par la proposition I.31 des *Éléments*, qui utilise I.23, et qui requiert pas moins de dix tracés de cercles !
3. Le point d est enfin construit comme intersection de ces deux dernières droites. Il faut pour cela démontrer qu'elles sont convergentes. Ici Von Plato utilise l'axiome de séparation suivant :

   $$\vdash \text{Con}(l, m) \rightarrow \vdash \text{Con}(l, n) \vee \text{Con}(m, n)$$

   qui dit qu'une troisième ligne coupe au moins une de deux lignes convergentes. Comme les lignes $par(ln(a, b), c)$ et $ln(a, c)$ sont convergentes en c, selon cet axiome la ligne parallèle à $ln(a, c)$ passant par b est forcément convergente avec la première, puisqu'elle ne l'est pas avec la seconde.

57. Von Plato utilise la notation abrégée suivante, qui vaut pour toute propriété $\vdash \text{Ppt}(x.y, z) \equiv \vdash \text{Ppt}(x, z) \wedge \text{Ppt}(y, z)$.

4. Les propriétés demandées concernant les points et les lignes ainsi obtenus doivent être vérifiés.

Cette preuve est tout entière contenue dans le contexte final, où on retrouve les quatre moments de la construction :

$$(a, b, c : \text{Point}, \tag{1.1}$$
$$\text{ln}(a, b), \text{ln}(a, c) : \text{Line}, \tag{2.1}$$
$$\text{par}(\text{ln}(a, b), c), \text{par}(\text{ln}(a, c), b) : \text{Line}, \tag{2.2}$$
$$d \equiv_{def} \text{pt}(\text{par}(ln(a, b), c), \text{par}(ln(a, c), b)) : \text{Point}; \tag{1.2}$$
$$\vdash \text{DiPt}(a, b), \vdash \text{DiPt}(c, d), \vdash \text{DiPt}(a.b, c.d); \tag{3}$$
$$\vdash \text{Inc}(a.b, \text{ln}(a, b)), \vdash \text{Inc}(c.d, \text{par}(\text{ln}(a, b), c)); \tag{4}$$
$$\vdash \text{Inc}(a.c, \text{ln}(a, c)), \vdash \text{Inc}(b.d, \text{par}(\text{ln}(a, c), b)); \tag{5}$$
$$\vdash \text{Par}(\text{ln}(a, b), \text{par}(\text{ln}(a, b), c)), \vdash \text{Par}(\text{ln}(a, c), \text{par}(\text{ln}(a, c), b)); \tag{6}$$
$$\vdash \text{DiLn}(\text{ln}(a, b), \text{par}(\text{ln}(a, b), c)), \vdash \text{DiLn}(\text{ln}(a, c), \text{par}(\text{ln}(a, c), b))). \tag{7}$$

On peut vérifier que toutes les expressions de ce contexte correspondent ligne à ligne à la spécification donnée plus haut, et qu'elles sont relatives aux éléments de départs a, b, c. Par souci de brieveté, nous avons introduit le terme d, mais elle peut être remplacée partout par le terme de droite de l'équation 1.2. Les preuves de ces assertions concernant des relations entre ces objets, simplement introduites par $\vdash$ (qui veut dire « qui peut être prouvé »), sont omises par souci de brieveté, mais elles découlent des axiomes posés par Von Plato, ainsi qu'on l'a vu pour le cas de la convergente des deux droites parallèles.

Dans cette nouvelle lecture, l'objet $y = B(x) : C(x, y)$ apparaît dans toute sa complexité : il n'est pas seulement la procédure qui construit l'objet spécifié par le problème, mais celle qui le construit *à partir* d'un contexte initial, et qui l'exhibe dans un contexte final où les propriétés nécessaires à sa vérification sont immédiatement lisibles.

Cet objet final peut se présenter aussi bien comme un théorème que comme un problème, selon qu'on choisit de mettre en avant une figure ou une relation (par exemple que les lignes $par(ln(a, b)$ et $par(ln(a, c)$ sont simplement convergentes, et d'en demander la preuve ou pas.

Ici apparaîtrait l'identité profonde des théorèmes et des problèmes dans l'interprétation de Mäenpää et Von Plato : qu'il s'agisse d'une preuve ou d'une construction, il s'agit de transformer un contexte initial en un contexte final, qui construit

de nouveaux objets à partir de ceux donnés initialement, que ces objets soient des objets élémentaires (points, triangles, etc.), des grandeurs, ou des preuves de relations entre ces objets ou ces grandeurs. Comme le dit lui-même Von Plato :

> De cette construction du parallélogramme, une manière générale de regarder les solutions des problèmes géométriques est la suivante : les solutions sont des correspondances entre un *contexte initial* (*data context*) du problème et son *contexte final* (*goal context*). Nous pouvons donner des expressions explicites de ces correspondances en théorie des types, comme fonctions entre ces contextes[58].

C'est bien ainsi qu'il faudrait donc se représenter toutes les preuves de la géométrie, qu'ils s'agissent de théorèmes ou de problèmes : comme des fonctions de transformations de contextes géométriques, qui comprennent tracés, observations, et mesures.

§ 154. Insuffisances de cette interprétation

La formalisation de Mäenpää et Von Plato n'est pas entièrement satisfaisante. D'abord, l'unification des objets géométriques (points, lignes, cercles) et de leur propriétés (distinction, séparation, convergence, etc.) sous un seul titre appelé « type » apparaît comme assez artificielle. Elle permet certes de présenter formellement les opérations de construction et de démonstration dans le cadre unifié du seul formalisme de la déduction naturelle, mais la convergence s'arrête là. Les propriétés sont des types clairement marqués par des $\vdash$, leurs objets sont toujours omis, et leur dérivation s'effectue selon les règles classiques de la logique des propositions, à partir d'axiomes. Or les très nombreux axiomes introduits par Von Plato ne figurent pas en géométrie euclidienne, qui n'a que des définitions, des postulats et des notions communes.

Par ailleurs, propositions et objets géométriques restent distingués implicitement dans la formulation même de la notion de problème comme $\forall x, \exists(y : B(x)) : C(x, y)$, où C est une proposition et B un objet géométrique. Si les types étaient vraiment traités de manière équivalente, cette distinction tomberait, et rien n'empêcherait de représenter les types complexes B et C par un seul type qui les agrégerait, comme $D(x) \equiv_{def} \exists(y : B(x)) : C(x, y)$.

C'est en fait bien à cela qu'aboutit Von Plato avec sa notion de *contexte*, qui est cependant insatisfaisante. Importée de la logique, elle présente l'objet géométrique complexe que construit le mathématicien comme une liste désordonnée et

58. (Von Plato 1995, p. 188).

incomplète d'objets et de propositions. Cette liste est désordonnée, puisqu'on aurait pu aussi bien regrouper, comme le fait d'ailleurs Von Plato, les points avec leurs propriétés, les lignes avec les leurs, et garder les relations d'incidence entre points et lignes à la fin. Cette liste est également incomplète, car bien d'autres propriétés pourraient être déduites de celles qui sont présentées, comme $\vdash \mathrm{Apt}(d, n)$ ou $\vdash \mathrm{Apt}(d, l)$. La fragmentation de l'objet géométrique construit en une multiplicité de types qu'il faut ensuite réunir dans l'unité artificielle d'un contexte ne traduit pas l'unité de l'investigation.

Ensuite, on ne voit pas très bien ce qui justifie l'introduction des propositions. Lorsqu'on traduit l'hypothèse de départ d'un énoncé géométrique, par exemple *« Soient a et b deux points distincts »*, par la position de deux objets a, b d'un type Point et l'assertion qu'un type paramétrique $\vdash \mathrm{Dipt}(a, b)$ est valide, cette assertion doit, en toute rigueur, avoir elle aussi une preuve ou un objet. Cependant on ne dit pas lequel il pourrait être. Il semble s'agir d'un acte de « lecture » des propriétés de l'objet constitué des deux points, mais Von Plato ne nous explique pas quels sont ces actes élémentaires.

On pourrait proposer d'introduire des actes élémentaires d'observation, comme par exemple

$$\mathrm{obsdi}(a, b) : \mathrm{DiPt}(a, b)$$

désignerait un acte d'observation du fait que a et b sont bien deux points distincts. Cependant un tel acte ne peut valoir comme preuve en géométrie euclidienne. Il est bien *évident*, par exemple, que l'orthocentre et le barycentre d'un triangle sont en général distincts, mais ce n'est pas faire de la géométrie que s'arrêter à ce constat sans le démontrer. On répondra : les actes canoniques $\mathrm{DiPt}(a, b)$ ne sont pas des observations, mais des conditions – par exemple ils pourraient être construits à partir d'un segment (a, b). Cependant dans la logique de Von Plato, ce sont les segments qui sont construits à partir de deux points distincts, et non l'inverse. On pourrait considérer les segments comme les objets originaires de notre logique, et définir les points comme des extrémités des segments. C'est la solution choisie par Naibo et Panza, cependant elle a le défaut d'inverser la priorité intuitive qu'on attribue en général aux points sur les segments, qui se voit dans l'ordre des définitions donné par Euclide lui-même. Une autre solution, enfin, est de faire un acte élémentaire du *choix* d'un point distinct d'un autre. Mais dans ce cas, ce qu'on construit n'est pas cette relation mais le point lui-même.

Enfin, Mäenpää et Von Plato ne traitent pas des propriétés essentielles des figures que sont les grandeurs (distances, angles, aires) sur lesquelles opèrent un calcul algébrique, une relation de congruence et une relation d'ordre. Les grandeurs semblent ainsi se comporter comme des types *distincts* des figures, mais ni leurs règles d'introduction, ni leurs règles d'usage ne sont définies.

Ces réflexions nous amènent au constat négatif suivant : dans la formalisation de Mäenpää et Von Plato, les preuves des théorèmes ne sont pas des cas particuliers de problèmes géométriques, sinon au sens très superficiel qu'on peut trouver un formalisme logique unifiant leur présentation. Pour la même raison, les propriétés des figures ne sont pas des types au même titre que peuvent l'être le point, la ligne ou le cercle. Elles concernent plutôt l'égalité de figures (« ce point, construit de telle manière, est *le même* que celui-ci, construit d'une autre »), l'ordre de points les uns relativement aux autres, et la grandeur comparée (de lignes, d'angles, de figures fermées). Mais de tout cela, la formalisation étudiée ne dit rien.

9.3 Une nouvelle interprétation constructive de la géométrie

Dans cette section nous esquissons une nouvelle interprétation constructive de la géométrie, qui complète de manière significative notre interprétation de la nature commune des règles du jugement et de l'action, qui se fondait sur la complémentarité des notions de *preuve* ou *construction* d'une part, et de *proposition* ou d'*objet* de l'autre, les uns étant les actes qui produisent ou rendent manifestes les autres. Nous montrons qu'on ne peut interpréter de cette manière la complémentarité des contenus constructifs et théoriques de la géométrie.

Nous commençons par mettre en avant le caractère problématique des *propriétés* par lesquelles les figures de la géométrie sont spécifiées, en particulier les grandeurs (distances, angles, aires) qui jouent un rôle essentiel dans l'armature logique des *Éléments* (§ 155). L'étude détaillée d'un théorème (I.13) nous montre que le discours sur les grandeurs procède par des actes de « lecture » de la figure qui permet de dégager des relations les concernant, qui peuvent ensuite être transformées grâce aux règles d'inférence que constituent les notions communes (§ 156). Puis, l'étude d'une séquence de théorèmes découlant de I.13 montre que leur objet est de rendre licite des actes de lecture de plus en plus sophistiqués, de la même

manière que les problèmes apprennent à construire des objets de plus en plus complexes (§ 157). Cela signifie-t-il que la géométrie serait bien sous-tendue par un dualisme fondamental entre actes constructifs et actes contemplatifs ? Nous avançons la thèse que, bien au contraire, son objet consiste dans l'étude de l'interdépendance de ces formes différentes de jugement, qui sont toutes deux constructives (§ 158). Notre dernier développement avance la thèse que cette situation est bien plus représentative des pratiques humaines complexes que celle de l'arithmétique, ce qui nous permet de distinguer l'identité des règles du jugement et de sa transformation, qui est fondamentale, de celle des règles de l'observation et de l'action au sein d'une même activité, qui est contingente et en général invalide. (§ 159).

§ 155. Questions et pistes

Partons de l'idée centrale que la géométrie a pour objets principaux les figures – points, segments et cercles – qu'elle peut construire et composer dans le plan. On dispose de méthodes de construction élémentaires pour ces objets. Pour les segments et les cercles, il s'agit des postulats 1 à 3. Quant aux points, leurs actes élémentaires sont d'être *marqués* à des situations particulières, qui peuvent être entièrement déterminées (intersection de deux droites convergentes), finitairement déterminées (il y a deux intersections de deux cercles incidents et non tangents) ou infiniment indéterminées (un point quelconque sur un segment ou un cercle, ou encore distinct de tout autre objet déjà construit). L'indétermination de ces points représente les entrées du problème, vis-à-vis desquelles ce dernier se comporte comme une fonction. Par là nous faisons une première proposition qui s'éloigne de la théorie de Mäenpää et Von Plato, en ce que nous considérons possible de marquer un point *distinct* d'une figure donnée, et en particulier d'un autre point.

Ces constructions initiales peuvent être composées entre elles pour former des méthodes de construction d'objets élémentaires (comme tracer la ligne parallèle à une autre, passant par un point donné) ou de figures complexes, comme un parallélogramme : c'est l'objet des *problèmes*. Nous donnons en annexe B la forme exacte de ces principes constructifs d'expression de la géométrie euclidienne, rédigés dans le langage Coq. Les réflexions qui suivent ont également vocation, dans un travail ultérieur, à trouver leur expression exacte dans ce langage.

Or deux difficultés apparaissent aussitôt : d'une part, on ne peut pas composer arbitrairement les constructions, et certaines séquences d'instructions peuvent être fautives. Par exemple on ne peut pas demander de marquer deux points distincts

à l'intersection de deux droites (puisqu'il n'en existe qu'un seul), ou de marquer l'intersection de deux cercles dont les centres sont plus éloignés que la somme de leurs rayons (puisqu'il n'y en a pas). Il semble y avoir des propriétés objectives de l'espace euclidien qui interdisent certaines constructions.

D'autre part, comment spécifie-t-on des problèmes *intéressants* ? il ne semble pas possible, comme pour les entiers naturels, de procéder à la définition par récurrence de propriétés et de fonctions intéressantes comme **pair**() ou **double**(), ainsi que nous l'avons fait plus haut. On imagine au mieux parler d'intersections et de parallélisme de figures entre elles, mais il ne semble pas qu'on puisse faire grand chose avec cela.

Lorsqu'on regarde de plus près *ce dont parlent* les *Éléments*, on s'aperçoit qu'il s'agit de *relations* entre figures. Il y a ainsi des relations de coïncidence (si trois segments égaux, partant chacun d'un point d'un cercle, ont une extrémité commune, alors *cette extrémité est le centre du cercle*, III.9) de distinction (si deux cercles se coupent alors ils n'ont pas même centre, III.5), d'intersection (deux cercles se coupent en deux points au plus, III.10), de parallélisme (si une transversale à deux droites fait leurs angles opposés égaux, alors ces deux droites sont parallèles, I.27), de situation relative (la corde entre deux points d'un cercle est intérieure à ce cercle, III.2).

Cependant, les relations les plus présentes dans les *Éléments*, qui forment l'essentiel de son armature conceptuelle, sont les comparaisons de *grandeurs* entre figures, principalement les distances entre points (ou longueurs de segments), les angles et les aires. Il n'existe presque pas de proposition qui n'y fasse appel, soit en termes de spécification (construire un triangle *équilatéral*, I.1) soit en termes de prémisses (des triangles qui ont la même base et sont égaux ont leur sommet sur la même parallèle, I.39). Enfin, même lorsque nulle grandeur n'est mentionnée dans le libellé d'un théorème, comme dans la proposition I.30 : deux lignes parallèles à une troisième le sont entre elles, la démonstration, elle, y fait appel (dans ce cas, Euclide utilise la propriété d'égalité des angles opposés). Les grandeurs servent donc, en particulier, à indiquer quelles constructions sont possibles ou impossibles, notamment concernant les intersections de figures. Elles sont donc une partie essentielle de l'armature logique des *Éléments*.

Or ce que sont les grandeurs est problématique dans notre perspective. Les relations de coïncidence ou de distinction entre figures peuvent se laisser interpréter, à la rigueur, comme la forme élémentaire de jugement $a = b$ qu'admet la théorie

des types. Les relations d'intersection, de parallélisme, ou de situation peuvent être interprétées, quant à elles, en termes de possibilité ou d'impossibilité de constructions de points ou de lignes. Mais les relations entre grandeurs semblent échapper à ces interprétations. Deux angles situés à deux endroits « lointains » d'une figure peuvent être dits égaux en vertu de raisonnements algébriques qui se détournent de la construction et semblent s'en abstraire.

Que sont donc toutes ces relations, et en particulier les grandeurs ? Si elles sont des types, quelles sont leurs règles d'introduction et d'utilisation ? Et surtout, comment comprendre qu'elles *contraignent* les constructions de figures, que nous pensions être les types primitifs de la géométrie ?

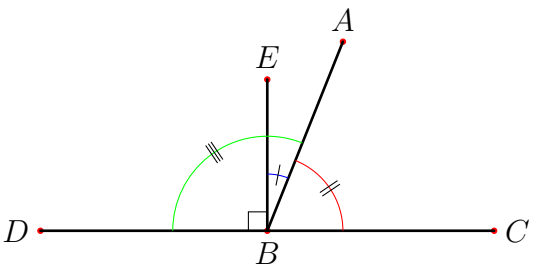
Il est tentant ici de donner raison à Platon, et d'interpréter ces propriétés comme des objets à part entière, caractéristiques de l'espace, accessibles par de purs actes d'observation – par des actes proprement « théoriques ». Les théorèmes, en particulier, joueraient le rôle essentiel de les mettre au jour. Cela amènerait à reconnaître qu'il existe bien deux formes d'actes intellectuels, de construction et d'observation, et donc également, des règles distinctes pour chacune de ces formes. Les analyses qui suivent confirment cette hypothèse ; cependant, loin de remettre en cause notre interprétation constructive de la géométrie, elle la renforce et l'enrichit.

§ 156. Un théorème angulaire

Afin de répondre à ces questions, il est utile d'étudier en détail un des premiers théorèmes des *Éléments*, la proposition I.13, qui a l'avantage de ne faire appel qu'aux notions communes et à une construction intermédiaire (voir figure 9.2) :

Théorème. *Si une droite est posée sur une autre droite, elle forme soit deux angles droits, soit des angles dont la somme est égale à deux angles droits.*

FIGURE 9.2 – Théorème I.13 des *Éléments* : somme des angles supplémentaires



Voici le texte de la démonstration d'Euclide :

Démonstration. Soit une droite quelconque AB posée sur une droite CD, et qui forme les angles CBA et ABD. Je dis que les angles CBA et ABD sont deux angles droits ou que leur somme est égale à deux angles droits.

1. Si l'angle CBA est égal à l'angle ABD, alors ce sont deux angles droits. (définition 10)
2. Si ce n'est pas le cas, il faut tracer BE à partir du point B en formant un angle droit avec CD. (I.11) Les angles CBE et EBD sont donc deux angles droits.
3. Puisque l'angle CBE est égal à la somme des deux angles CBA et ABE, ajoutez l'angle EBD à chacun d'eux. La somme des angles CBE et EBD est égale à la somme des trois angles CBA, ABE et EBD. (notion commune 2)
4. Pareillement, Puisque l'angle DBA est égal à la somme des deux angles DBE et EBA, ajoutez l'angle ABC à chacun d'eux. La somme des angles DBA et ABC est égale à la somme des trois angles DBE, EBA et ABC. (notion commune 2)
5. Mais la somme des angles CBE et EBD s'est aussi avérée égale à la somme des trois mêmes angles, et les choses qui sont égales à une même chose sont aussi égales entre elles, donc la somme des angles CBE et EBD est aussi égale à la somme des angles DBA et ABC. Mais les angles CBE et EBD sont deux angles droits, donc la somme des angles DBA et ABC est aussi égale à deux angles droits. (notion commune 1)

Par conséquent, si une droite est posée sur une droite, elle forme soit deux angles droits, soit des angles dont la somme est égale à deux angles droits. □

Si on met de côté le cas trivial où AB est perpendiculaire à CD (point n°1), la démonstration comprend trois formes d'actes : d'abord, la construction d'une ligne auxiliaire BE (point n°2), puis la « lecture » d'une relation additive entre les angles ABC, EBA et EBC d'une part, et entre DBE, EBA et DBA d'autre part (points n°3 et 4), et enfin des inférences tirées de ces relations grâce aux notions communes (points 3 à 5). On peut détailler ces deux dernières formes d'actes sous la forme du raisonnement suivant (ici $2d$ vaut « deux angles droits », et NC =

notion commune) :

$$CBA + ABE = CBE \qquad \text{(lecture n°3)}$$
$$CBA + ABE + EBD = CBE + EBD \qquad \text{(inférence n°3, NC 2)}$$
$$ABE + EBD = ABD \qquad \text{(lecture n°4)}$$
$$ABE + EBD + CBA = CBA + ABD \qquad \text{(inférence n°4, NC 2)}$$
$$CBA + ABD = CBE + EBD = 2d \qquad \text{(inférence n°5, de 3 \& 4, NC 1)}$$

Quelle est donc la nature de ces inférences et de ces actes de « lecture » de la figure ? Concernant les inférences, il est clair qu'il s'agit d'un calcul propositionnel dont les notions communes forment les règles d'inférence. Ce calcul propositionnel comprend quelques formes élémentaires, concernant l'égalité ou l'inégalité de deux angles, ou la relation additive entre trois angles ou plus. On se trouve donc dans le cas prévu par la théorie des types, où des propositions sont prises elles-mêmes comme objets d'un type soumis à des règles spécifiques[59].

Reste donc à expliquer la nature des actes de « lecture » que nous trouvons dans les points n°3 et 4 de la démonstration. Ils concernent la relation additive entre trois angles, à savoir qu'étant donnés trois segments de même sommet, la somme des deux angles formés par le segment intérieur avec les deux segments extérieurs est égale à l'angle formé par ces deux derniers (dans le diagramme, par exemple BA est le segment intérieur aux deux segments extérieurs BE et BC).

La possibilité d'une telle lecture n'est pas justifiée par Euclide, ainsi que cela a été remarqué[60] ; elle semble s'ancrer dans le principe évident que le tout est la somme de ses parties. Elle est en tous les cas tout à fait formalisable en théorie des types, comme une règle de formation d'une proposition à partir de trois lignes de même sommet. Il s'agit donc d'un acte à part entière, d'une nature entièrement distincte de celle des actes de construction, qui permet en quelque sorte de former des objets propositionnels à partir de certaines figures.

Nous donnons un fragment de cette théorie des grandeurs, écrit en Coq, en annexe B de ce travail. Il est important ici de remarquer que, comme les constructions élémentaires, les « actes de lecture » sont réduits au strict minimum par Euclide. Ainsi l'égalité entre angles est réduite au cas des angles droits, et fait l'objet du postulat 4 (« tous les angles droits sont égaux entre eux »), ce qui montre son im-

59. Voir plus haut, § 148.
60. Voir par exemple le commentaire de (Joyce 1998).

portance. Toutes les autres égalités entre angles résultent d'inférences utilisant les notions communes.

Le théorème I.13 permet donc de mettre en rapport deux actes d'un type différent : celui, d'une part, de construire une ligne incidente à une autre, et celui, d'autre part, déduit par des actes de lecture et d'inférence, de juger que les angles formés par cette ligne sont supplémentaires (*i.e.* d'une somme égale à deux droits). Il forme donc une sorte de « couplage de significations » reliant deux jugements, l'un concernant une construction, et l'autre des grandeurs. Le théorème I.14 vient d'ailleurs clôturer cette « capsule de sens », en montrant la proposition réciproque : si la somme de deux angles adjacents sont égaux à deux droits, alors les deux segments extérieurs de ces angles sont situés sur la même ligne.

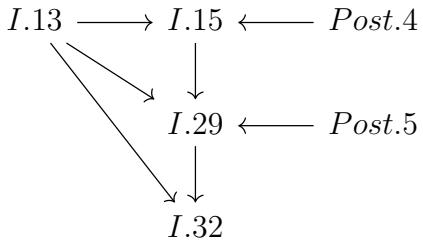
L'usage des théorèmes est double. D'une part, il permet de développer d'autres théorèmes associant figures et relations de grandeurs ; d'autre part, il permet de démontrer la possibilité ou l'impossibilité de certaines constructions. Concernant le second point, nous nous contentons ici de prendre l'exemple de la proposition I.27, qui montre que si une transversale à deux lignes fait des angles alternés égaux, alors *il n'est pas possible* de marquer un point d'intersection entre elles (*i.e.* elles sont parallèles). Nous allons développer plus en détail le premier point, puisque nous cherchons à élucider la nature des théorèmes.

§ 157. Une séquence de théorèmes

Nous nous intéressons à présent à la progression logique, schématisée en figure 9.3, de certains théorèmes découlant de I.13. En partant par le bas, I.32, qui établit le célèbre théorème que la somme des angles d'un triangle est égale à deux droits, ne fait appel qu'à des constructions auxiliaires, des notions communes et aux théorèmes I.13 et I.29. I.29, à son tour, qui établit les propriétés angulaires relatives à des parallèles coupées par une transversale, ne fait appel qu'aux notions communes, au postulat 5 et aux théorèmes I.13 et I.15. Enfin I.15, qui établit l'égalité d'angles opposés, dépend seulement des notions communes, du postulat 4 et du théorème I.13 (les notions communes et les constructions auxiliaires ne sont pas représentées sur le diagramme). On voit ici que la proposition I.13, que nous venons d'étudier, joue un rôle essentiel dans les inférences angulaires du livre I des *Éléments*.

Nous allons passer ces théorèmes rapidement en revue, en résumant leurs démonstrations par souci de concision.

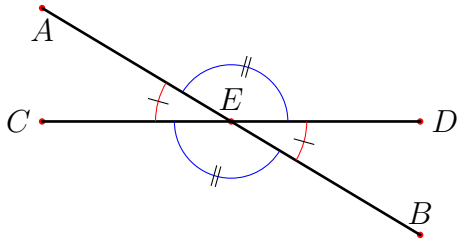
FIGURE 9.3 – Un des développements du théorème I.13



Le théorème I.15 illustre très bien comment de nouvelles relations angulaires sont mises au jour à partir de la figure du théorème I.13, légèrement complétée :

Théorème. *Si deux droites se coupent mutuellement, elles font les angles au sommet égaux entre eux.*

FIGURE 9.4 – Théorème I.15 des *Éléments* : égalité des angles opposés



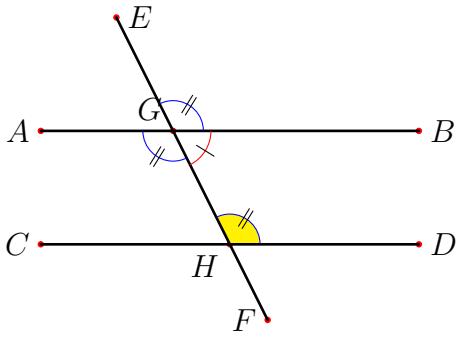
La preuve opère de manière identique par des actions de lecture et d'inférence. Seulement cette fois-ci les actes de lecture portent sur la configuration déjà étudiée dans la proposition I.13, où un segment est seulement posé sur un autre (sans le couper). Il s'agit d'une part de AE qui est posée sur CD en E, et d'autre part de DE qui est posée sur AB en E. Il est très important de bien noter la lecture active qui est ici requise : il s'agit à chaque fois de *choisir* une sous-partie de la figure complète, en « oubliant » le segment EB d'abord, puis le segment EC ensuite. Cela permet d'extraire de ces sous-parties la propriété déjà démontrée par I.13, qui devient disponible pour le calcul propositionnel :

(1)	$CEA + AED = 2d$	(lecture n°1 prop. 13)
(2)	$AED + DEB = 2d$	(lecture n°2 prop. 13)
(3)	$CEA + AED = AED + DEB$	(Post. 4 + NC 1)
(4)	$CEA = DEB.$	(NC 3)

Le théorème I.29, quant à lui, enrichit la figure I.15 de la manière suivante (voir figure 9.5) :

Théorème. *Une droite tombant sur des droites parallèles rend les angles alternés égaux entre eux, l'angle extérieur égal à l'angle intérieur et opposé, et la somme des angles intérieurs d'un même côté égale à deux angles droits.*

FIGURE 9.5 – Théorème I.29 des *Éléments* : égalité des angles d'une transversale avec deux lignes parallèles



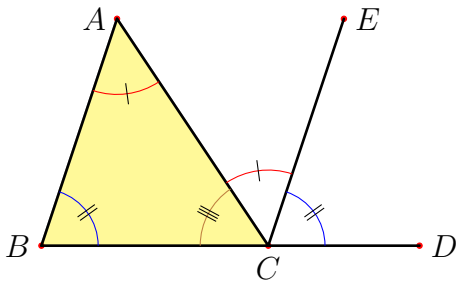
Nous omettons ici entièrement la démonstration, qui procède comme pour I.15, par des actes de lecture sélective et par des inférences suivant les notions communes. C'est l'angle GHD qui est ici central : le théorème démontre son égalité d'une part avec l'angle alterné AGH, d'autre part avec l'angle opposé intérieur BGE, et enfin que sa somme avec l'angle intérieur de même côté HGB est égale à deux angles droits. Deux actes de lecture sont ici mobilisés, l'un de la configuration de type I.13 formée par le segment GH posé sur le segment AB en G, et l'autre de celle de type I.15 formée par le croisement des segments AB et EH en G.

On arrive enfin au théorème I.32, qui établit la célèbre égalité de la somme des angles d'un triangle à deux droits (voir figure 9.6).

Théorème. *Dans un triangle quelconque, si l'un des côtés est produit, l'angle extérieur est égal à la somme des deux angles intérieurs et opposés, et la somme des*

trois angles intérieurs du triangle est égale à deux angles droits.

FIGURE 9.6 – Théorème I.32 des *Éléments* : égalité des angles intérieurs d'un triangle à deux droits



C'est grâce à la construction de CE, la ligne parallèle à BA, qu'on peut démontrer cette égalité, mais il est à remarquer que cette construction ne joue aucun rôle dans le résultat final. Il est donc possible qu'une propriété d'une figure ne puisse être rendue visible que par « extraction » d'une figure plus complexe, ce qui donne toute sa subtilité à la géométrie euclidienne. Comme dans les exemples précédents, la réflexion procède par quatre *lectures sélectives* successives de la figure, qui permet de n'en considérer que des sous-parties. Une configuration de type I.29 est ainsi lue sur les lignes AB, AC, CE, puis une autre sur les lignes AB, CE, BD. Une lecture « additive » élémentaire est ensuite réalisée sur les trois segments CA, CE, CD. Enfin, une configuration de type I.13 est lue sur le segment AC posé sur le segment BD en C. La réflexion extrait de toutes ces lectures des égalités angulaires sur lesquelles la logique arithmétique des notions communes peut être appliquée. Le raisonnement algébrique peut alors être déployé aux étapes 4 et 5 :

(1)	$CAB = ACE$	(lecture n°1 prop. 29)
(2)	$ABC = ECD$	(lecture n°2 prop. 29)
(3)	$ACD = ACE + ECD$	lecture n°3 « additive »
	$ACD = CAB + ABC$	(1+2)
(4)	$BCA + ACD = BCA + CAB + ABC$	(NC 2)
(5)	$2d = BCA + CAB + ABC.$	(lecture n°4 prop. 13, NC 1)

On voit donc cette séquence de théorèmes enrichir progressivement par la démonstration les actes de lecture d'égalités angulaires de sorte qu'ils puissent s'appliquer

à des configurations de plus en en plus riches ; ils deviennent aussi moins « évidents », si l'évidence est liée à leur adjacence – comme le suggère l'acte de lecture initial qui se contente d'additionner les angles adjacents et qui est, il faut bien le dire, postulé. Ces actes de lecture se comportent donc exactement comme les actes de construction qui, partant de postulats initiaux, se composent les uns les autres en actes de plus en plus complexes.

§ 158. Le jugement et l'action

Cette conclusion semble donc bel et bien militer en faveur d'actes « théoriques » (que nous avons ici appelés actes de « lecture ») et qui se composent selon leur logique propre au long des *Éléments*, de manière indépendante et parallèle à celle des constructions de figure. De ce fait, la dualité entre théorèmes et problèmes semble devoir être maintenue. Cela ne veut-il donc pas dire qu'il y a bien une distinction radicale entre deux domaines de règles, les unes portant sur des actes de construction, et les autres sur les propriétés des choses ?

En fait les choses ne se passent pas tout à fait ainsi, et il n'y a pas une vraie symétrie entre constructions et observations, entre figures et grandeurs, car les grandeurs sont, pour ainsi dire, des « objets fantômes ». Nous voulons dire par là que les actes qui permettraient à proprement parler de les exhiber, les *mesures*, ne sont pas des actes admis par le géomètre ; celui-ci en parle et les compare relativement les unes aux autres, et peut même en tirer des ratios, mais il ne les réalise jamais. Les grandeurs (distances, angles, aires) ne sont pas des objets accessibles directement, et n'apparaissent qu'au sein de propositions qui les mettent en relation.

Deux interprétations sont possibles, qui prennent acte de cette déréalisation. La première tend à les voir comme des relations abstraites et formelles entre figures, qui n'ont plus grand chose à voir avec la mesure proprement dite. C'est la grandeur de l'axiomatisation de Tarski d'avoir montré que toutes les relations entre figures – et partant tous les théorèmes et problèmes – étaient exprimables à l'aide de deux seules relations formelles (*i.e.* définies par les seuls axiomes régissant leur manipulation, sans référence à leur interprétation) entre points, la relation de congruence et d'intermédiarité[61].

Cependant, dans cette axiomatique, la plupart des théorèmes concernant les grandeurs, comme ceux que nous venons d'étudier, perdent leur sens, car il n'est

61. Voir plus haut, § 150.

plus possible en particulier d'y parler de leur *somme*. Au contraire, chez Euclide, la possibilité d'additionner les grandeurs montre qu'elles ont une réalité intrinsèque.

Une seconde interprétation serait possible, qui pousserait à son terme la refondation constructive de la géométrie euclidienne. Il est envisageable de « simuler » tout acte de mesure par des constructions géométriques. Les mesures d'angles et d'aires peuvent en effet être ramenées à des mesures de distance, et les jugements concernant les égalités et inégalités de celles-ci peuvent à leur tour être traduites par des jugements concernant des coïncidences entre points, à l'aide de constructions auxiliaires s'appuyant sur le postulat 3 pour tracer des cercles de congruence et le postulat 2 pour situer des points relativement à ces cercles. Dans cette perspective, les théorèmes de la géométrie pourraient tous être ramenés à la forme de jugement $a = b$ admise par la théorie des types, où a et b seraient des points ou plus généralement des figures.

Cependant il ne nous semble pas intéressant de poursuivre une telle orthodoxie formelle. Comme l'axiomatisation de Tarski, elle manquerait l'esprit de la géométrie euclidienne, qui garde un ancrage fort dans des *pratiques* humaines, et en particulier celle de la mesure de l'espace. Le livre II, qui traite d'algèbre géométrique (par exemple, II.2 montre la distributivité de la multiplication sur l'addition $x(y_1 + y_2) = xy_1 + xy_2$ à l'aide de raisonnements sur les rectangles) montre qu'une des finalités de la géométrie pour les Grecs était le calcul *in fine* de grandeurs concrètes servant sans doute des besoins pratiques.

Aussi notre interprétation – qui nous semble conforme à l'opinion reçue – est tout simplement que si les grandeurs apparaissent de manière si indirecte dans les *Éléments*, c'est que la mesure n'est pas le domaine de la géométrie, mais celui de l'arpentage. D'une certaine manière, un des objectifs essentiels de la géométrie est de *calculer exactement* les mesures et de les *prévoir*, ce pour quoi elle s'interdit *justement* de les mesurer. Les actes de mesure sont bel et bien présents dans la géométrie – mais ils sont ce dont on ne parle jamais, car ils sont de l'ordre de la vérification empirique, sur laquelle le géomètre, du faîte de sa certitude, n'abaisse guère son regard.

Mais alors, d'où viennent ces « actes de lecture » évoqués plus haut, et qu'est-ce qui les légitime ? Il faut ici revenir aux postulats, qu'on associe trop souvent, notamment les trois premiers, aux seuls actes de construction. En fait ils prodiguent également des *droits* à réaliser des inférences élémentaires concernant les grandeurs. Ainsi, le cercle est défini comme l'ensemble des points équidistants du centre

(définition 15), et on postule *par ailleurs* qu'on sait le construire (postulat 3). Ayant construit un cercle, et marqué deux points dessus, il est donc licite de juger que leurs distances respectives au centre sont égales, *par définition*. De l'extension d'un segment AB à partir de B vers un point C (postulat 2) on a le droit d'inférer que B est situé entre A et C, ou que la distance AC est plus grande que AB. Le postulat 4, qui dit simplement que tous les angles droits sont égaux, permet d'inférer la congruence de tels angles même s'ils ne sont pas adjacents et partant, grâce aux raisonnements algébriques des notions communes, de tous autres angles. Ce postulat est nécessaire car la définition de l'angle droit (déf. I.10) est elle-même relative : est droit l'angle d'une ligne posée sur une autre lorsqu'il est égal à son angle adjacent. Le postulat 4 assure donc qu'il n'y a qu'une seule manière de tracer une telle ligne sur un même côté de la base. Quant à l'additivité des angles adjacents qui est cruciale pour le théorème I.13 étudié plus haut, il est sans aucun doute l'un de ces rares postulats implicites qu'on trouve dans les *Éléments*.

Nous ne cherchons pas ici à être exhaustifs. Ce que nous voulons montrer, c'est que les actes élémentaires de construction et de d'observation sont dans une interdépendance réciproque ; les postulats et les définitions permettent ensemble de définir des *couplages de signification* élémentaires, qui disent tout à la fois quelles figures il faut savoir construire, et quelles relations on peut y lire. À partir de ces couplages élémentaires, la géométrie développe des couplages de plus en plus sophistiqués, montrant comment construire des figures de plus en plus complexes et comment inférer d'elles des relations de plus en plus subtiles. L'alternance des problèmes et des théorèmes est donc une modalité essentielle de la réflexion du géomètre.

S'il y a donc un primat des théorèmes sur les problèmes, comme le suggère la tradition, c'est qu'ils fournissent l'armature logique de la géométrie. Les propositions *encadrent* les constructions, en fixant leurs spécifications mais aussi en validant leur possibilité. Un programme de construction, pris tout seul, ne dit rien et ne garantit rien. Il n'indique pas quelles seront les propriétés de la figure finale. Il n'est pas même certain d'aboutir. Par exemple, une instruction pourrait demander de prendre l'intersection de deux cercles, alors que la distance séparant leurs centres serait supérieure à la somme de leurs rayons. La géométrie commence lorsqu'on vise à établir des jugements certains concernant les figures construites, et à s'assurer que les constructions aboutissent toujours. Elle formule des conditions à la réalisation de certains actes (notamment la construction d'intersections), dé-

crit les propriétés des constructions élémentaires (la ligne et le cercle), et les règles d'inférence de certaines propriétés à d'autres. Elle peut, dans cette perspective, être vue comme une logique de Hoare[62], qui s'assure de la correction des constructions proposées. Elle permet de dériver rigoureusement des propriétés initiales de la figure ses propriétés finales, et de vérifier que ces dernières ne contiennent pas de contradictions.

§ 159. L'action complexe et les types interdépendants

L'interprétation de la géométrie euclidienne que nous avons dégagée, inspirée de la théorie des types, enrichit de manière significative la réponse dégagée plus tôt concernant la nature commune des règles du jugement et de l'action.

Il y a en effet une différence essentielle entre la géométrie et l'arithmétique, d'un point de vue constructif : les entiers naturels se laissent définir de manière indépendante, c'est-à-dire hors de toute contrainte imposée par un autre type ; ils sont d'ailleurs, à bien des égards, au fondement de tous les autres. Ce n'est pas le cas en géométrie : figure et grandeur sont des types mutuellement dépendants, dont les relations, décrites par l'ensemble définitions / postulats, sont encapsulées dans les figures élémentaires de la ligne et du cercle. On y trouve de manière indissociable aussi bien les constructions élémentaires qu'on peut faire que les relations de grandeurs qu'on peut y lire – par exemple que par deux points passe une seule droite, que les parallèles ont égaux leurs angles alternés avec une transversale, etc.

Cette complexité de la géométrie, où plusieurs types doivent coopérer pour capturer un aspect du réel – pour rendre compte des règles par lesquelles nos actions peuvent réussir dans le monde – est bien le cas général qu'il faut considérer du point de vue pratique, plutôt que celui des entiers naturels.

En arithmétique en effet, les fonctions comme **double**() et les prédicats comme **pair**() sont établis sur le même schéma récursif et sont à ce titre homogènes. Il n'y a qu'une seule « capsule de sens » élémentaire, celle qui associe la définition inductive de toute expression à son calcul récursif, et qui permet de montrer la vérité d'une proposition comme $\forall n \in \mathbb{N}, \mathbf{pair}(\mathbf{double}(n))$ sur la seule base de la définition de ses termes.

La dualité entre règles du jugement et règles de l'action provient donc du fait qu'en général notre relation avec le réel a plus de similarité avec notre expérience

62. Voir plus haut, § 138.

de l'espace qu'avec celle de la manipulation de séries homogènes d'objets. Nous voulons dire par là que notre relation avec les choses ne se laisse modéliser que de manière exceptionnelle et très locale par des algorithmes manipulant seulement des entiers naturels. Le plus souvent, on y trouve plutôt une complémentarité entre des actes productifs, qui peuvent échouer, et des actes d'évaluation, capables de détecter cet échec et parfois de le prévenir – comme il y a complémentarité entre la main et l'œil du chirurgien. Il ne s'agit pas là d'une dualité entre action et observation, entre production et contemplation, entre pratique et théorie ; les actes d'évaluation sont tout autant des productifs d'objets que les actes de construction ; simplement, ils agissent en général sur un autre plan ontique, tout comme un superviseur derrière un moniteur de commande est actif d'une autre manière que le technicien intervenant sur le terrain.

Cette dualité n'est pas nécessaire ; elle n'existe pas dans le cas des raisonnements portant sur les entiers naturels, comme nous l'avons vu ; et on peut à l'opposé imaginer aussi des contextes où *plus de deux types* sont nécessaires pour rendre compte de la complexité des règles en jeu. D'une certaine manière, la multiplicité des grandeurs géométriques (distances, angles, aires, arcs, etc.) pointe déjà vers cette possibilité.

La logique dynamique, inspirée de la logique de Hoare[63], est une manière de modéliser un rapport possible entre deux types de pratiques, où l'une apparaît comme « statique » (le domaine du jugement concernant des états du monde) et l'autre « dynamique » (le domaine de l'action modifiant ces états du monde et donc ces jugements eux-mêmes). Mais pour voir que ces notions de *statique* et *dynamique* sont relatives et au fond métaphoriques, il suffit de prendre un exemple où l'état du monde n'est pas naïvement modélisé comme une « photo » de choses immobiles, mais comme le fonctionnement régulier de processus – par exemple de processus industriels qui eux-mêmes requièrent des actions, des jugements et des décisions permanentes de la part des agents impliqués. Modifier cet état de choses, c'est-à-dire introduire une nouvelle norme industrielle, comme changer la cadence ou l'ordre d'un processus, va requérir de ces agents qu'ils modifient tous leurs comportements – non seulement leurs actions, mais leurs procédures de décision elles-mêmes pour s'adapter à cette nouvelle situation et recréer un nouvel équilibre dynamique.

La programmation, en particulier, tient bien plus de cette complexité géomé-

63. Voir plus haut, § 138.

trique que de la simplicité arithmétique à laquelle on l'associe souvent. Un programme, en tant qu'il agit sur l'espace de la mémoire, est une construction dont la correction dépend de la structure de cet espace. Tout programme un tant soit peu complexe comporte plusieurs structures de données, et emploie plusieurs types. Il s'agit là d'une différence notable avec les algorithmes, qui souvent ne manipulent de manière principale qu'un seul type de données. Ce fait explique aussi la pertinence de concepts comme la programmation objet* ou les types abstraits de données* qui associent étroitement les structures de données à des opérations qui leurs sont associées.

Notre question concernant l'unité de l'action et du jugement sort donc enrichie de la considération de la géométrie. Elle apparaît comme plus complexe que ne le suggérait initialement une interprétation uniforme de la théorie des types. Une réponse en deux moments est ainsi requise :

1) Au niveau d'un domaine (ou encore *type*) donné, action et jugement apparaissent bien comme les deux faces d'une même pièce. Ce sont les mêmes règles qui les régissent tous deux, comme on le voit bien dans le cas de l'arithmétique. Juger selon une règle est une forme d'acte qui transforme un contexte de connaissance. Réciproquement toute action se fait dans l'intention de transformer une situation, qui implique qu'on puisse juger son effectivité. Ainsi calculer des nombres n'est pas moins une action qu'empiler des briques. Et empiler des briques conformément à un plan requiert autant de jugement qu'établir ce plan lui-même. La théorie des types interprète le jugement comme l'assertion qu'un problème est résolu conformément à sa spécification, ou que telle méthode résout systématiquement un problème paramétrique, ou encore que deux problèmes sont équivalents. Le jugement se rapporte toujours à une action qui elle-même s'achève toujours dans un jugement.

2) La plupart des activités humaines complexes passent cependant par l'interaction de plusieurs domaines d'action, chacun doté de ses types de résultats propres. En particulier, une dichotomie courante – mais nullement absolue – a lieu entre les domaines considérés comme productifs d'effets intéressants dans le monde et ceux qui vérifient ou contrôlent ces effets ; et *dans ces cas de figures*, des règles différentes peuvent sembler régir le domaine de l'action et le domaine du jugement ; mais il ne s'agit là que d'une forme de cette division du travail intellectuel qui se développe à partir des Temps modernes[64]. Nous revenons sur cette dicho-

64. Voir plus haut, 7.3.

tomie en conclusion de ce chapitre. Il n'existe donc pas une distinction d'essence entre acte et jugement, entre domaine de l'action et domaine de la contemplation, entre savoir pratique et théorique ; mais une multiplicité indéfinie de pratiques interdépendantes qui possèdent leurs propres normes d'action et de jugement.

9.4 La systématicité

Dans cette section nous dégageons de nos réflexions sur la géométrie euclidienne une tentative de définition *conjointe* de la générativité et de la systématicité. On se rappelle en effet que ces deux notions semblent antithétiques. Quand on essaie d'organiser un ensemble de règles descriptives d'une situation, les méthodes qui viennent à l'esprit sont celle de la division – on distingue un cas générique en cas spécifiques puis en cas de plus en plus particuliers – ou celle de la combinaison : on se donne d'emblée un ensemble fini de caractéristiques descriptives de la situation, et on obtient toutes les situations possibles par combinaison. Ces deux méthodes sont connues des Grecs, et organisent les réductions en art jusqu'au XVII^{e} siècle.

Le problème de ces méthodes est qu'elles ne rendent pas compte de la dynamique propre à l'action ou au jugement. Rien de neuf ne peut survenir dans les situations quadrillées par les méthodes arborescentes ou combinatoires, puisqu'elles sont en nombre fini – rien donc ne peut être fait ou jugé qui n'ait pas été déjà jugé à l'avance. En d'autres termes, il ne peut y avoir de *vrais problèmes* à résoudre, mais seulement des recettes infaillibles à appliquer. C'est en partie pour cette raison, on s'en souvient, qu'Aristote était resté dubitatif quant à une possible systématisation de l'action humaine.

Nous tentons ici d'analyser et de généraliser ce que nous apprend la géométrie concernant la possibilité de systèmes génératifs de règles (permettant d'exprimer un nombre indéfini de problèmes). Il faut pour cela élucider ce que peut être la résolution systématique de problème de manière la plus large possible – qui n'exige pas que la solution soit donnée à l'avance dans le système. Cette élucidation passe par celles des notions d'*exactitude* et d'*exhaustivité* (§ 160), qui dévoilent à leur tour le caractère central des structures opératoires *inductives*. Notre développement suivant s'attache à clarifier cette idée, en revenant sur la structure récursive des entiers naturels, dont l'importance tient selon nous à ce qu'elle est la forme la plus simple d'une systématicité infinitaire (§ 161). Ces résultats nous permettent de

répondre enfin à une question qui nous occupe depuis le commencement de notre réflexion, concernant la possibilité d'une science ou d'une logique universelle des opérations. Nous illustrons comment les principes dégagés sont à l'œuvre dans les langages de programmation classiques, dits programmation impératives*, puis dans un modèle classique de la calculabilité, les systèmes de Post (§ 162).

§ 160. La systématicité

L'esprit systématique, nous dit Kant, est un fait de la raison qui examine la somme des règles conçues par l'entendement et entreprend de les organiser ; il procède donc d'une démarche réflexive, de second ordre relativement à l'entendement. L'entendement sait seulement concevoir des règles, c'est-à-dire chercher ce qu'il faut faire ou ce qu'il faut juger dans telle ou telle situation. Il sait aussi les appliquer, c'est-à-dire les reconnaître à travers les situations rencontrées. Cependant il ne va pas au-delà.

Notre thèse est que l'organisation des règles par la raison n'a pas seulement une valeur « bibliothécaire », comme le soutient le philosophe des mathématiques Wang[65], mais qu'elle permet la *résolution systématique de problèmes*. et que c'est donc dans cette approche spécifique de l'investigation qu'il faut chercher le sens de la *systématicité*.

L'esprit systématique, avons-nous dit, requiert exactitude et exhaustivité[66]. L'*exactitude* signifie que l'investigation se déroule strictement selon des règles (il n'y a pas d'« improvisation »), et que toutes les règles utilisées ont été préalablement définies – on ne peut en définir de nouvelles au cours de l'investigation. Cette seconde condition n'est pas naturelle. C'est en réfléchissant à un problème qu'on s'aperçoit en général de la nécessité et de la validité d'une règle, et on l'énonce alors en passant. Nous avons vu un tel exemple avec la règle que la somme de deux angles adjacents en une ligne intérieure est égale à l'angle constitué par les deux lignes extérieures[67].

L'esprit systématique ne se satisfait pas d'une telle désinvolture, car il sait que des règles énoncées en passant peuvent recéler des erreurs subtiles. Il exige que toutes les règles soient consignées à part et préalablement, de sorte qu'elles puissent être l'objet d'un contrôle séparé, et que l'investigation se limite strictement à leur

65. Sur ce point, voir notre discussion en fin de chapitre, § 165.
66. Voir plus haut, § 143.
67. Voir plus haut, § 156.

usage – cela implique également qu'elles soient en nombre fini – un point sur lequel nous reviendrons[68]. On retrouve ici l'idée de la troisième division du travail identifiée autour de la figure de l'ingénieur, qui doit *utiliser* la théorie *construite* par le savant[69]. Nous avions déjà noté que cette caractéristique était celle précisément mise en avant par Ryle dans son analyse de la systématicité[70]. La géométrie euclidienne donne le modèle quasi-parfait de cette division du travail, puisque toutes les opérations de construction et de lecture (à part quelques oublis mineurs comme la règle énoncée ci-dessus) sont exactement définies. *Exactitude* est d'ailleurs le terme qu'emploie Bos pour désigner l'objet principal des recherches géométriques du Grand Siècle, *i.e.* ces constructions non réductibles aux opérations de la règle et du compas, et aux opérations complémentaires qu'il fallait admettre pour les réaliser[71]. L'exactitude, en ce sens, n'est pas du tout une affaire de *précision*, mais de la conformité ou non à des règles opératoires données ; elle ne souffre pas le plus ou le moins ; elle est, pour reprendre l'expression de Ryle, *procrustéenne*[72].

La générativité de la géométrie (le fait qu'elle permette de décrire des situations et des problèmes en nombre indéfini) tient au fait que les opérations de construction élémentaires qu'elle définit (marquer des points, tirer des lignes, tracer des cercles) peuvent être *composées* entre elles afin d'engendrer des figures de plus en plus riches : lorsqu'on dispose de deux points initiaux, on peut tirer un segment les reliant et construire deux cercles ; mais on peut alors également marquer les deux intersections entre ces deux cercles, et également celles que chacun d'entre eux a avec le segment prolongé en ses deux extrémités, etc. Bien plus, de nouveaux points quelconques peuvent toujours être marqués sur ces figures ou hors d'elles, ce qui fait que la géométrie n'est jamais limitée par un matériau initial fini.

Deux principes complémentaires sont ici a l'œuvre. D'abord, les règles ne disent pas qu'il faut aller de A à B puis de B à C – où A, B et C sont statiques et donnés par avance, mais de A à $f(A)$ puis à $g(f(A))$, puis encore, par exemple, à $f(g(f(A)))$, à $f(f(g(f(A))))$, etc. et cela de manière indéfinie. Les opérateurs transforment une situation en une autre qui reste disponible pour que les opérateurs s'y appliquent à nouveau. Nous appelons cela le *principe de composition* au sens de la composition de fonctions mathématiques : une

68. Voir plus loin, § 194.
69. Voir plus haut, § 118.
70. Voir plus haut, § 92.
71. Voir plus haut, § 124.
72. Voir plus haut, § 18.

séquence ordonnée d'actions compose une situation à partir de la suite des transformations unitaires opérées par chacune.

Ensuite, ces transformations unitaires sont locales, c'est-à-dire que leur résultat ne dépend pas de l'ensemble de la configuration initiale. Une règle ne s'applique qu'à un nombre fini des caractéristiques d'une situation, qu'elle « filtre » pour ainsi dire afin de déterminer l'action. Nous l'avons bien vu dans les démonstrations des théorèmes d'Euclide : c'est en considérant seulement une partie des éléments d'une figure que l'investigateur est capable d'y reconnaître une configuration déjà rencontrée, pour laquelle il dispose d'une règle. Nous appelons ce second principe *principe de localité*. Sans lui, il n'y aurait pas de vraie générativité, puisqu'il faudrait définir l'action des opérateurs sur chaque configuration possible. Il n'est en fait pas si évident que cela : dans un schéma arborescent, c'est l'ensemble de la situation qui détermine la règle, de ses caractéristiques les plus générales aux plus particulières. Aussi de tels schémas sont-ils peu pratiques pour traiter des cas de polymorphisme, où une même différence pertinente apparaît dans des cas très éloignés spécifiquement ou génériquement.

Ces deux principes, pris ensemble, expliquent en partie la générativité de la géométrie, *i.e.* comment un nombre de règles finies peuvent engendrer des situations indéfiniment diverses. Chaque construction d'un point, d'une ligne ou d'un cercle est une fonction qui agit différemment d'une configuration à l'autre, comme le même mouvement d'un cavalier aux échecs, sur les mêmes cases, a cependant un sens qui varie en fonction de la situation des autres pièces sur l'échiquier, ou comme l'ajout de $+1$ à 5 ou à 55 donne évidemment des résultats différents. Ce dernier exemple permet de noter que ces deux principes caractérisent également les deux règles qui engendrent les entiers naturels en nombre indéfini ; la composition est évidente dans la construction $\mathbf{succ}(\mathbf{succ}(\ldots(\mathbf{succ}(\mathbf{0}))))$; quant à la localité elle se voit en ce que $\mathbf{succ}()$ ne s'applique qu'à son antécédent immédiat.

On peut reconnaître ici ce que nous avons appelé plus haut des *structures inductives*, et c'est donc bien le caractère inductif des constructions géométriques qui explique leur générativité[73]. Plus généralement, la notion d'induction est centrale à la définition des types, comme nous allons le voir au développement suivant.

Avant cela, il nous faut analyser la notion d'*exhaustivité*. Cette idée est d'abord intuitive. Du moment qu'on propose la solution d'un problème, il faut s'assurer de son périmètre d'application, c'est-à-dire des conditions dans lesquelles elle reste

73. Voir plus haut, § 151 et, de manière un peu plus détaillée plus loin, § 165.

pertinente – ce qui correspond au concept de *fonction partielle* en mathématiques. Il se peut par ailleurs que la solution puisse être aisément adaptée à des cas particuliers ou à des exceptions ; dans ce cas, on réunit toutes ces variantes dans la solution en précisant leurs cas d'application, et on obtient une portée plus générale de la solution. On trouve ici la seconde caractéristique de la systématicité que nous avions dégagé de la réflexion de Ryle, celle de la « décontextualisation » des savoirs théoriques[74]. Il nous faut à présent préciser cette idée.

L'exhaustivité est relative à une classe homogène de problèmes dont on veut donner une solution unique, et donc aux différents sens qu'on peut donner à ces notions d'*homogénéité* et d'*unicité*. La géométrie en donne d'une certaine manière le modèle, puisque l'espace euclidien est homogène et que toute procédure indiquée s'applique presque toujours de manière uniforme en tous points de l'espace. Il se peut que, concernant des constructions, des conditions d'existence soient nécessaires, par exemple pour l'intersection de deux cercles, mais celles-ci peuvent être incluses dans l'énoncé de manière à sauvegarder l'homogénéité de la procédure. Par ailleurs il se peut aussi parfois que la procédure diffère en fonction de la configuration de départ ; par exemple le théorème I.35 des *Éléments* requiert trois démonstrations légèrement différentes en fonction de la position d'un point relativement à deux autres dans l'énoncé. Cependant ces « accidents topologiques » sont rares et ne sont pas au cœur de la géométrie euclidienne.

Le problème de l'exhaustivité se pose donc lorsqu'il y a des *cas* ou des *exceptions* relativement complexes à traiter dans une situation par ailleurs homogène. Si ces cas ou exceptions sont en un nombre fini A, B, C, D, E, etc., il suffit de les énumérer et de spécifier la procédure adéquate pour chacun. Cependant, étant donné le principe de localité, il faut envisager le cas où plusieurs exceptions locales se composent entre elles (par exemple AB, BD, etc.) et requièrent une nouvelle règle, spécifique à leur combinaison. On comprend que de telles complications peuvent se composer à leur tour entre elles (D, CD, ABE, CDE, etc.) et la question se pose alors de savoir jusqu'où on doit accepter de décrire comme « homogène » une situation repliée de multiples fois en cas et en sous-cas, et balafrée d'accidents de toutes parts aux combinaisons elles-mêmes sporadiques. L'exemple de Deep Blue, le programme d'échecs, a montré que les limites de l'exhaustivité pouvaient être *en droit* poussées très loin, en tous les cas au-delà de celles où l'énoncé d'une telle

74. Voir plus haut, § 92.

règle restait en pratique pertinent pour des personnes humaines[75].

Afin de préciser cela, on peut imaginer des cas encore plus complexes d'exceptions qu'on jugerait cependant acceptables : il se peut par exemple que la règle à appliquer dépende de la répétition de n marques A spécifiques dans la situation, ou du fait qu'il y ait deux fois plus de marques A que de marques B, ou encore que le nombre de marques A, B, et C soient dans une relation arithmétique entre eux, etc. On voit où nous voulons en venir : le problème, avant d'appliquer la bonne règle, est de *savoir reconnaître selon des règles* à quelle situation on a affaire. Ce travail là doit lui-même être exact si on veut préserver l'exactitude de l'ensemble de la procédure, et donc être lui-même décrit comme une composition d'opérateurs définis exactement sur un espace donné. Autrement dit, l'espace des cas possibles (ou exceptions) doit lui-même pouvoir être parcouru de manière inductive (donc : par un nombre fini de règles) afin qu'on puisse rester dans le cas d'une résolution systématique de problèmes.

§ 161. La récursivité, ou la systématicité infinitaire

La notion de structure inductive est donc centrale dans l'idée d'une systématicité générative. La définition inductive des entiers naturels donnée plus haut est le modèle le plus simple d'une systématicité infinitaire. Or ce n'est pas seulement leur définition qui est inductive, mais également celle de leurs différents usages. On peut remarquer en effet que nous avions également appliqué cette même méthode pour définir des fonctions comme **double**(n) ou **demi**(n), des propriétés, comme **pair**(n) ou encore des preuves, comme celle de **pair**(**double**(n)).

Il y a là un schéma général. Pour définir un usage des entiers naturels, il suffit de savoir ce qu'il faut faire pour **0** puis comment transformer ce qu'on sait faire pour x en ce qu'il faut faire pour **succ**(x). Par exemple, l'addition d'un entier quelconque à un entier m donné peut être définie comme suit :

$$m + \mathbf{0} \equiv_{def} m$$
$$m + \mathbf{succ}(n) \equiv_{def} \mathbf{succ}(m + n)$$

Ou encore[76], la construction des listes de longueur n sur un ensemble A, les

75. Voir plus haut, § 71.
76. Exemple pris dans (Granström 2011, p. 73).

pré-conditions de ces assertions étant placées à leur droite :

$$[\ ] \in \mathbf{List}(A, \mathbf{0}) \qquad (A \text{ set})$$
$$[a, l] \in \mathbf{List}(A, \mathbf{succ}(n)) \qquad (a \in A, l \in \mathbf{List}(A, (n)))$$

Dans ce second exemple, on utilise les entiers naturels pour construire un nouveau type. Dans le premier, raisonner revient à faire du calcul, c'est-à-dire des substitutions de termes selon des règles jusqu'à ce qu'on aboutisse à des éléments déjà définis. l'addition revient à trouver un entier naturel de la forme **0** ou **succ**(**succ**(...(**succ**(**0**)))).

Ces expressions peuvent être ramenées à un schéma commun, spécifique à chaque type, qu'on appelle *opérateur d'élimination*[77], *constante non-canonique*[78], ou encore *récurseur*[79]. Ainsi, toute expression d'un ensemble C quelconque indexée sur les entiers naturels peut être définie à l'aide de l'opérateur **natrec** (*récurrence des entiers naturels*) calculant l'expression $c_n \in C$ pour tout $n \in \mathbb{N}$ à l'aide de deux arguments fixes :

1. Une valeur $c_0 \in C$ indiquant la valeur de l'expression pour **0**.
2. Une fonction $f : \mathbb{N} \times C \to C$ qui permet de calculer l'expression $c_{\mathbf{succ}(n)} \in C$ si on connaît $c_n \in C$ et $n \in C$.

On a alors :

$$\mathbf{natrec}(c_0, f)(0) = c_0 \in C$$
$$\mathbf{natrec}(c_0, f)(\mathbf{succ}(n)) = f(n, \mathbf{natrec}(c_0, f)(n)) \in C$$

Toutes les expressions utilisant les entiers naturels peuvent être formalisées grâce à cet opérateur. Par exemple, pour l'addition, il faut donner à **natrec** la valeur initiale m et la fonction $g(x, y) = \mathbf{succ}(y)$. On a alors :

$$\mathbf{natrec}(m, g)(0) = m$$
$$\mathbf{natrec}(m, g)(\mathbf{succ}(n)) = \mathbf{succ}(\mathbf{natrec}(m, g)(n))$$

La première ligne équivaut à la proposition $m + 0 = m$, et la seconde à $m + \mathbf{succ}(n) = \mathbf{succ}(m + n)$.

En théorie des types, ce schéma est central à la définition de tout type. La dualité des éléments canoniques, introduits par les constructeurs du type, et des

77. (Martin-Löf 1984, préface).
78. (Nordström, Petersson et J. M. Smith 1990).
79. (Univalent Foundations Program 2013).

éléments non-canoniques, introduits par les opérateurs d'induction, reflète deux mouvements possibles de la réflexion : le premier, celui de se donner un espace et de le *parcourir*, et le second *d'agir en fonction du cas rencontré*. Les opérateurs d'induction permettent de définir le second type de règles, celles qu'on évoque usuellement et qui ont la forme *condition* $\rightarrow$ *action*. Les premières sont caractéristiques d'une démarche systématique, celle de créer un espace des conditions *au sein desquelles* de telles règles trouveront leur détermination.

On voit ici la puissance que peut acquérir une telle résolution systématique de problème lorsqu'elle se dédouble ainsi, d'une part en une règle qui indique comment parcourir l'espace des conditions, d'autre part en une autre règle qui dit comment l'action doit se déformer relativement à ce parcours. Une telle solution n'a en effet plus besoin d'être entièrement énumérée selon toutes ses conditions possibles et exceptions pour être valide. Il suffit qu'on sache la générer le moment venu en fonction de la situation rencontrée, si celle-ci peut elle-même être décrite de manière inductive à partir d'une situation de référence pour laquelle la règle est déjà connue.

Or les espaces finis de conditions, binaires ou plus généralement les énumérations k-aires, peuvent également être dotés d'un opérateur non-canonique, qui se contente de sélectionner et d'appliquer la i-ème règle d'une liste ($i < 2$ ou $i < k$ respectivement)[80]. Aussi peut-on dire que tout espace de conditions, fini ou infini, doit être défini de manière inductive.

La possibilité de représenter la solution de tout problème de manière systématique n'est pas évidente. On l'a vu avec le jeu d'échecs : il se peut que l'espace pertinent des conditions soit trop *enchevêtré* pour que cela soit réalisable pratiquement ; on peut aussi imaginer que dans certains cas cet espace de conditions ne puisse être énuméré : il s'agit là d'une question classique de la philosophie des sciences cognitives, celle de l'hyper-computation[81]. Nous préférons, de notre côté, retourner la question et remarquer que la systématicité n'est pas une condition nécessaire à la résolution de problème, mais une de ses manières possibles.

Un investigateur humain « non systématique » se contente en effet, la plupart du temps, de composer une solution de son problème à partir du matériau empirique des conditions et des opérations possibles qu'il sait reconnaître dans l'expérience, et qu'il peut adapter marginalement à la situation présente. Une telle

80. (Nordström, Petersson et J. M. Smith 1990, p. 41-45).
81. Voir plus haut, § 22.

démarche est doublement approximative, en ce que les règles qu'elle mobilise ne sont pas nécessairement extraites d'un corpus de règles vérifiées, et en ce qu'elle ne peut déterminer à l'avance la portée de sa solution relativement à des exceptions possibles.

Le point essentiel concerne ici la notion d'*exhaustivité*. On l'a vu, elle est relative à la portée que l'on souhaite donner à la solution d'un problème ; par exemple aux échecs, il n'est pas évident que *la même* procédure de réflexion doive résoudre les problèmes de milieu de partie aussi bien que ceux de fin de partie, ceux où on est en position de force aussi bien que ceux où on est en position de faiblesse, ceux où l'on dispose de sa reine aussi bien que ceux où on l'a perdue, etc. Le joueur humain aura tendance, dans tous ces cas de figure, *à réfléchir différemment*. On peut donc très bien limiter la recherche d'exhaustivité à une définition assez étroite du problème en jeu.

À l'inverse, on peut, grâce aux méthodes statistiques de l'apprentissage profond* trouver des solutions globales à des problèmes extrêmement complexes, comme le jeu d'échecs et le jeu de go, la reconnaissance d'images, etc. De telles résolutions de problèmes ne sont pas systématiques en ce que le principe de localité n'y est pas respecté : chaque opérateur (par exemple un neurone individuel dans un réseau de neurones*) dépend en droit de l'ensemble des conditions de la situation[82].

D'une certaine manière, donc, entreprendre de résoudre *systématiquement* un problème relève d'un choix méthodologique, qui n'est pas le seul possible, même lorsqu'on mobilise des procédés calculatoires. Ce choix consiste à représenter le problème par un espace de conditions engendré par un nombre fini de règles, qu'un nombre fini de règles localement opératoires peut transformer ; et à rechercher une solution de ce problème par la seule composition de ces dernières. Le choix de la systématicité offre des avantages certains quant à l'automatisation, au contrôle et à la traçabilité des solutions[83], mais il est important pour notre réflexion de ne pas le confondre avec le concept même d'investigation et de procédure qui sont bien plus vastes. Nous y revenons dans la conclusion de notre travail.

82. Nous développons ce point plus loin, § 239.
83. Voir plus haut, 5.4.

§ 162. La science des opérations

Il est à présent possible d'esquisser les contours de cette « science des opérations » ou de cette « mathématique de l'action » que nous cherchons depuis le début de notre réflexion. Une telle mathématique ne cherche pas à cataloguer ou à abstraire l'ensemble des actions possibles des personnes humaines. Elle s'occupe seulement d'organiser des espaces de conditions et d'actions données par l'investigation dans l'analyse d'un problème pour qu'on puisse y formuler les règles d'action qui le résolvent systématiquement, c'est-à-dire exactement et exhaustivement, pour toutes les conditions sur lesquelles il est défini.

Cette mathématique, on l'a vu ci-dessus avec l'exemple de la géométrie, nous semble fondée sur trois principes d'organisation des règles, de *composition*, de *localité* et d'*induction*. Ces trois principes, s'ils correspondent assez bien à l'idée que nous nous faisons aujourd'hui du changement naturel, *font violence* à la logique habituelle des actions humaines, telle que l'avait décrite Aristote sous le titre général de *prudence*. Le problème du cadrage[84], qui porte sur la difficulté d'identifier de manière exhaustive les significations pertinentes d'une situation problématique, remet en effet en cause le principe d'induction. Il y a des situations que les règles semblent incapables de décrire d'une manière définitive, car des éléments *nouveaux* ou des exceptions *imprévisibles* doivent parfois être pris en compte pour former la décision ; il n'y a donc pas de procédure permettant de parcourir exhaustivement l'ensemble des conditions possibles du problème, et il n'est donc pas non plus possible de formuler une solution systématique de ce dernier. La mise en œuvre systématique de l'action, telle que la conçoit la logique industrielle, requiert que des cadres techniques soient mis en place afin que le problème du cadrage soit résorbé du mieux possible, et que ces trois principes puissent être appliqués.

Ces trois principes sont assez directement visibles dans la théorie des langages de programmation. Cette théorie procède souvent par l'étude d'un ensemble minimal de quelques instructions, qu'on appelle un *langage-jouet* (*toy language*). L'idée est de réduire la complexité de l'étude, tout en gardant néanmoins l'universalité caractéristique des ordinateurs. Ainsi dans leur manuel *Software Foundations*, Pierce et al. présentent le langage *Imp* (pour *Imperative language*) constitué, de manière très classique, des cinq commandes suivantes[85] :

84. Voir plus haut, § 9.
85. (Pierce et al. 2023), ch. *Imp*.

CODE 9.2 – Définition du langage *Imp* en Coq

```
1  Inductive com : Type :=
2  | Skip
3  | Asgn (x : string) (a : expr)
4  | Seq (c1 c2 : com)
5  | If (b : bexp) (c1 c2 : com)
6  | While (b : bexp) (c : com).
```

La première commande, `Skip` ne fait rien (*skip* = passer son tour, ignorer) – elle est d'une utilité simplement technique. La seconde commande implémente le principe de localité, puisque `Asgn (x:string) (a: expr)` (*assigner*) modifie localement la situation : dans le langage *Imp* – et en général dans la plupart des modèles de machines – la situation est modélisée par une « mémoire » qui peut prendre différentes valeurs. Ici `x` désigne un emplacement reconnu de la mémoire auquel une valeur `a` va être *assignée*, elle-même fonction de certains éléments de la mémoire conjoints dans une *expression* (`expr`). Dans *Imp*, il s'agit d'expressions arithmétiques, mais toute forme de fonction est ici possible. La commande `Asgn` transforme donc de manière *homogène* la mémoire, au sens où l'expression `expr` agit de manière similaire sur toutes ses configurations possibles.

La commande `Seq` traduit assez directement le principe de composition. Ordonner en effet que les deux opérations `c1` et `c2` soient exécutées l'une après l'autre signifie que l'état initial du système s sera transformé en $f_2(f_1(s))$, si f_1 et f_2 sont les transformations ordonnées respectivement par `c1` et `c2`.

Les deux dernières commandes, `If` et `While`, relèvent du principe d'induction, l'un pour les cas en nombre défini et l'autre pour les cas en nombre indéfini. Ce lien n'est pas évident et il faut donc le développer.

La signification de `If` est d'exécuter `c1` si `b` est vraie, et `c2` sinon, et elle implémente donc l'idée d'une gestion de cas ou d'exceptions. Comme nous l'avons dit plus haut, on peut voir là l'opérateur d'induction (ou encore de la constante non-canonique[86]) d'un ensemble à deux éléments dont le cas général est l'induction sur un ensemble fini. Nordström, Petersson et J. M. Smith appellent `case ... of` l'opérateur non-canonique qui parcourt un ensemble fini à la recherche du cas pertinent à appliquer[87]. On peut obtenir cette instruction par une série de `If` enchâssés.

86. Voir plus haut, § 161.
87. (NORDSTRÖM, PETERSSON et J. M. SMITH 1990, p. 41-45).

Enfin, la signification de la commande `While (b : bexp) (c: com)` est de répéter le sous-programme `c` tant que l'expression booléenne `b` est vraie. Afin que cette répétition s'arrête il faut donc que `b` devienne fausse à un moment, ce qui ne peut se faire que si dans le sous-programme `c` les déterminants de `b` sont en mesure d'être modifiés. Cette propriété va donc être vraie n fois (le nombre de répétitions de `c`) avant de devenir finalement fausse. Il y a donc, sous-jacente, une induction indexée par les entiers naturels qui est opérée par `c`.

Un tel langage-jouet permet de « tout » programmer, au sens où il permet d'émuler le comportement de tout ordinateur classique. On voit en quoi consiste cette mathématique des opérations : elle ne s'occupe pas de définir les opérations concrètes elles-mêmes qui sont en jeu : elle ne fait que les esquisser par l'opération `Asgn` qui désigne une transformation locale opérant sur un matériau donné (principe de localité) ; son véritable objet est de composer ces règles entre elles et de gérer leurs conditions, qu'elles soient en nombre fini ou indéfini. Ensemble, les trois méta-opérations `Seq, If, While` permettent de rendre l'action systématique.

On voit également ces trois principes à l'œuvre dans les systèmes de Post qui sont, on le rappelle, au côtés du λ-calcul* et des machines de Turing, l'un des modèles théoriques les plus connus de la calculabilité.

Un système de Post n'est rien d'autre qu'une liste de règles disant chacune comment transformer un type de situation, décrite par une suite de marques symboliques, en une autre situation. L'exécution du système consiste à réitérer indéfiniment l'application de ces règles à cette phrase symbolique jusqu'à ce que plus aucune de ces règles ne s'y applique.

Plus précisément, une règle exprime seulement la réécriture d'un schéma de séquence symbolique (son antécédent) en un autre schéma (son conséquent) qui reprend les termes variables du premier en les déplaçant (un même terme pouvant être répété plusieurs fois) ou en les éliminant. Une règle de production s'écrit :

$$g_0 P_1 g_1 P_2 g_2 ... P_m g_m \rightarrow h_0 Q_1 h_1 Q_2 h_2 ... Q_n h_n$$

où les g et les h désignent des séquences de symboles fixées, les P, des séquences quelconques de symboles (« variables »), et chaque Q un élément de la liste de séquences $\{P_1, P_2, ..., P_m\}$. Une telle règle de production exprime exactement le principe de localité puisqu'elle réagit seulement à un schéma fini de caractères de la situation. Par exemple la règle :

```
1    aPbQc → aQdPc
```

se déclenche uniquement si la situation décrite est encadrée par les lettres `a`, `c` et contient la lettre `b` au milieu, éventuellement séparée de `a` et `c` par des suites quelconques de caractères `P` et `Q`. Dans ce cas, elle transforme la situation en remplaçant `b` par `d` et en inversant la position de `P` et `Q`. Cette règle pourrait par exemple être intéressante pour inverser deux blocs descriptifs d'une situation, comme la gauche et la droite si on se retourne, et garder la mémoire de son orientation, exprimée par `b` ou `d`.

Quant aux principes de composition et d'induction, ils sont adroitement entremêlés dans la réitération indéfinie du système de règles, qui peut être vue comme la composition, indéfinie et sans condition, d'une même fonction $f(f(f(..)))$. Ainsi, la commande `Seq` du langage *Imp* ci-dessus peut être modélisée par des règles chaînées les unes aux autres de la manière suivante :

```
1    c1 → c2
2    c2 → c3
3    c3 → c4
```

Dans ce cas, si la situation initiale est `c1`, le programme exécutera successivement les lignes 1, 2, 3. `If` est également très simple à écrire dans un système de Post. Si `b1` et `b2` sont deux valeurs booléennes inverses, et qu'il faut exécuter `c1` dans le premier cas, et `c2` dans le second, on écrit tout simplement :

```
1    b1  → c1
2    b2  → c2
```

Pour illustrer `While`, prenons l'exemple d'une procédure exécutant l'action `c` si la situation `b`, elle-même déclenchée par l'action `a`, n'a pas été rencontrée *plus de trois fois*, et sinon l'action `d`. Il peut s'agir, par exemple, d'une procédure disant à un garde de laisser passer les trois premières personnes qui se présentent, mais de bloquer toutes les autres. Dans ce cas, un compteur doit être incrémenté chaque fois que la situation `b` est atteinte pour savoir dans quel cas on se situe. Le compteur va être représenté par la répétition d'un symbole choisi, par exemple `*`, et dont la séquence va être placée à la suite de chaque situation. On a ainsi les règles suivantes, où `P` représente une séquence quelconque de `*`, éventuellement vide :

```
1    aP  → bP
2    b   → c*
```

```
3       b*     → c**
4       b**    → c***
5       b*** → d
6       cP     → aP
```

Le nombre de visites de `b`, marquée par `P`, est toujours transmise telle quelle d'une situation à une autre (voir lignes 1, 6), sauf dans le cas où l'état `b` est atteint, dans ce cas on lui ajoute une `*` (ligne 2 à 4). Lorsque la situation `b` est atteinte trois fois, on passe à `d` (ligne 5). Si la situation initiale est donc `a`, ce système de Post s'exécute donc selon la séquence suivante : 1 - 2 - 6 - 1 - 3 - 6 - 1 - 4 - 6 - 1 - 5. On a ici un exemple simple d'induction jusqu'à $n = 3$.

Il faut noter que les systèmes de Post, comme la commande `While` vue plus haut, sont plus permissifs qu'une induction stricte telle que la définit la théorie des types : ils peuvent en effet laisser des boucles opérer indéfiniment sans qu'une condition d'arrêt soit jamais rencontrée. Cette possibilité traduit bien ce qu'est la tâche même du programmeur, à savoir s'assurer que son espace de conditions est bien défini et que sa règle le parcourt d'une manière correcte.

9.5 L'action mathématique et l'action réelle

En quelle mesure ces résultats répondent-ils à notre questionnement général ? Ils permettent certes de nous faire une idée exacte de la systématicité, mais parlent-ils vraiment de ce qui nous intéresse – à savoir des actions et des pratiques humaines concrètes et de la manière dont elles sont reconfigurées par la réflexion systématique ? Nous examinons dans cette section les prétentions des mathématiciens intuitionnistes à étendre leur théorie à l'ensemble du champ de la connaissance, d'abord les quelques indications parcellaires de Martin-Löf lui-même (§ 163) puis celles plus développées de son élève Mäenpää (§ 164). Nous les trouvons insuffisantes, ce qui nous conduit à revenir à la perspective pragmatique évoquée au début de ce chapitre (§ 165). Celle-ci, au contraire, semble avoir quelque difficulté à concevoir la possibilité et l'importance d'une approche proprement systématique.

§ 163. L'oubli de la délibération

Martin-Löf défend explicitement l'idée que sa logique des problèmes n'est pas limitée à des actes et à des objets de connaissance, mais qu'elle pourrait valoir

pour tout type d'acte constructif :

> Les notions d'acte et d'objet, qui se sont révélées cruciales dans l'analyse de la notion de jugement ou de connaissance, aussi ambiguë soit-elle entre l'acte de juger ou de savoir, d'une part, et ce qui est jugé ou connu, d'autre part, c'est-à-dire l'objet de la connaissance, ces deux notions ne se limitent nullement aux actes de connaissance, car le concept d'acte peut être compris dans le sens le plus large possible. Dans la division traditionnelle, il comprend aussi bien des pensées que des paroles et des actes, c'est-à-dire des actes mentaux, des actes verbaux et des actes concrets, comme faire un gâteau ou frotter un sol ou quelque chose comme ça. […] Il s'avère que plusieurs des notions que j'ai déjà évoquées [concernant la logique du jugement] peuvent être élevées au niveau des actes et des objets en général. Il en est ainsi, en particulier, des notions de possibilité logique, d'actualité et de possibilité réelle : elles ont certainement un sens pour n'importe quel objet, pas seulement pour les objets de connaissance. En effet, un objet est logiquement possible s'il a été simplement posé ce que signifie de le faire (*if it merely has been laid down what is meant by doing it*), c'est-à-dire en accomplissant un acte avec cet objet, et un objet est réel s'il a été fait, c'est-à-dire si un acte a été accompli avec cet objet, et un objet est réellement possible s'il peut être fait, c'est-à-dire si un acte avec cet objet peut être accompli[88].

Martin-Löf nous invite ainsi lui-même à étendre ses thèses, au-delà des mathématiques, vers l'ensemble du domaine de la connaissance et même au-delà, vers l'ensemble de l'action humaine. Le titre d'un des ses articles, *A Path From Logics to Metaphysics*, « Un chemin de la logique vers la métaphysique », indique bien son ambition, et il définit souvent la logique comme « théorie générale de la connaissance[89] ».

Il n'a cependant pas publié de travaux portant sur ces questions. Mäenpää mentionne un cours de Martin-Löf évoquant l'épistémologie des sciences de la nature[90]. Dans les travaux accessibles, ses réflexions prennent la forme d'exemples d'évidence empirique, qui reviennent dans plusieurs textes. Nous en citons deux, tirés de (Martin-Löf 1994).

Le premier exemple est celui de savoir si le soleil brille. Dans ce cas, dit Martin-Löf, il suffit d'être dehors, ou de regarder par la fenêtre. Voir le ciel est l'acte élémentaire qui nous permet de savoir si le soleil brille. Ce critère est en partie arbitraire, car on pourrait exiger seulement de se fier aux bulletins météo. Cepen-

88. (Martin-Löf 1991, p. 145-146).
89. (Martin-Löf 1987, p. 419), (Martin-Löf [1983] 1996, p. 20).
90. (Mäenpää 1993, p. 35).

dant, s'il s'agit de retrouver l'évidence la plus immédiate d'un jour non voilé, voir le ciel est sans doute l'expérience élémentaire. On peut réaliser cette expérience soi-même : si je suis à l'intérieur d'une habitation, la procédure que je dois suivre est d'en sortir, ou d'aller à une fenêtre. Elle peut être également réalisée par un tiers digne de confiance, une autre personne, ou une machine. Dans tous ces cas, je dois élaborer et vérifier des procédures qui, exécutées, provoquent l'expérience élémentaire « voir le ciel ». Savoir que le soleil brille implique de réaliser, d'une manière ou d'une autre, cet acte d'évidence.

Le second exemple de Martin-Löf semble très similaire au premier. Il s'agit de savoir si la température est de 10°C. Dans ce cas, l'acte élémentaire correspondant à cette connaissance est de lire un thermomètre. L'intérêt de ce second exemple est de montrer que le savoir immédiat ne se réduit pas à des expériences perceptuelles immédiates du phénomène en question. Martin-Löf ne mentionne pas ici l'expérience d'avoir froid ou pas, mais une *mesure* de température. Toute mesure implique un protocole de mesure, et en général un instrument, ici le thermomètre. On pourrait certes demander d'auditer ce thermomètre afin de s'assurer de son bon fonctionnement, et dans ce cas ne devrait-on pas compter comme acte élémentaire le *couple* ⟨auditer le thermomètre, lire le thermomètre⟩ ? Un tel affinement est certainement possible. À notre avis, la naïveté des exemples de Martin-Löf est feinte. Car on pourrait rétorquer au demi-habile demandant la vérification du thermomètre de vérifier également la procédure de vérification, de mesurer les risques de dérèglement du dispositif entre le moment de l'audit et celui de la mesure effective, etc. Le point essentiel de ce second exemple nous semble donc être qu'une mesure est d'ores et déjà un objet *construit*, et que sa mise en évidence est toujours située dans un contexte complexe qui implique la manipulation d'objets plus élémentaires selon une procédure. L'expérience, hormis les cas simples comme le premier exemple ci-dessus, suppose un tel *dispositif* – Martin-Löf utilise le terme « complexe » (*complex*)[91]– au sein duquel peut être conduit un acte d'*acquisition* empirique du savoir. Mäenpää développe des remarques pragmatiques dans ce sens[92].

Au sein de ce dispositif, dont on suppose qu'il est correct (le thermomètre, la procédure, mes propres capacités mentales), dire qu'il fait 25°C lorsque le thermomètre indique 25°C est, selon Martin-Löf, un acte analytique, car la mesure

91. (Martin-Löf 1994, p. 90).
92. (Mäenpää 1993, p. 36).

du thermomètre décrit justement une preuve canonique de cette proposition, il en constitue la signification même. L'assertion de la proposition n'est synthétique que si elle est réalisée *hors de la présence du thermomètre*, car la preuve est manquante, et « vous devez aller au-delà de tout ce qui est contenu dans le jugement pour vous le rendre évident[93] ».

L'exemple du thermomètre montre que ce qui, dans une théorie donnée, est défini comme objet élémentaire ne dispose pas en général d'une évidence immédiate. Ce qui est déterminé comme « expérience immédiate » ne doit donc pas être compris dans le sens d'une doctrine sensualiste de la connaissance, qui voudrait la réduire à des *qualia* ultimes. Elle désigne plutôt ce qui est posé, ou postulé, comme actes et objets immédiats d'un système de connaissances, à partir desquels celui-ci peut se déployer en procédures discursives. Ainsi les objets immédiats de la physique quantique, ses fameux observables, comme le spin de l'électron, sont associés à des procédures extrêmement complexes de mesure, d'un point de vue pragmatique. Ce sont pourtant ces procédures qui doivent être tenues pour immédiates du point de vue du physicien, même s'il doit s'appuyer ici sur une infrastructure épistémique sophistiquée – savoirs et outils mécaniques, électromagnétiques, informatiques, etc. – lui permettant de réaliser ces actes. Dans un autre texte, Martin-Löf note que les termes *évidence* et *démonstration* sont métaphoriques de la vision, mais qu'on pourrait tout aussi bien les remplacer par ceux de *justification* et de *garantie*, où c'est la métaphore légale qui a cours[94].

Il y a certainement là des pistes de réflexion concernant l'épistémologie qui pourraient être développées, et qui semblent pointer vers une approche opérationnelle de l'expérimentation qui peut faire penser à Bridgman[95], c'est-à-dire qui définit le contenu de l'expérience à partir son protocole de réalisation.

Là n'est pas cependant l'objet de ce travail et nous nous demandons plus généralement si ces remarques peuvent suffire à expliquer l'effectivité de cette logique des opérations dans les nombreux domaines de l'activité humaine qu'elle pénètre aujourd'hui.

Il nous semble que cela n'est pas le cas. En effet, les exemples de Martin-Löf, malgré leur subtilité, ne nous disent pas vraiment grand-chose de ce qui nous intéresse, l'investigation. Observer le temps par la fenêtre ou relever la température

93. (Martin-Löf 1994, p. 88).
94. (Martin-Löf 1995, p. 188).
95. (Chang 2021).

au thermomètre sont deux protocoles, l'un issu de la vie courante, l'autre d'une très longue histoire du concept de *température*. Si on peut nier que le premier résulte d'une quelconque investigation, tant il nous semble naturel, il n'en est pas de même du second. Martin-Löf nous donne des procédures pour résoudre des problèmes ou encore, pour parler son langage, des types qu'il a déjà définis avec toutes leurs règles, mais il ne nous dit rien de la manière dont il est arrivé à ce résultat. C'est comme lorsqu'on présente la fonction **demi**(y) pour résoudre le problème $P3(y, x)$: cette fonction est certainement correcte, mais dans cette présentation il n'y a nulle trace de l'investigation. Toute la complexité de la délibération humaine est de bien poser les problèmes et de définir les chemins pour les résoudre, et non d'appliquer des procédures connues.

Il se pourrait ainsi qu'en décrivant trop directement le savoir pratique à l'aide de la théorie des types, on retombe, comme par mégarde, dans une vision « théorique » de la connaissance. Cette théorie nous présente certes des procédures qui expriment rigoureusement comment résoudre de manière effective un problème, mais cette présentation toute hiératique des étapes à suivre ne garde plus trace de l'investigation vivante qui l'a, peut-être difficilement, amenée au jour, et dont la description ne se confondrait certainement pas avec elle. En d'autres termes, nous risquons de manquer de voir, encore une fois, comment cette résolution systématique se relie à l'attitude réflexive de la délibération tendue vers un problème à résoudre. Représenter les pratiques humaines selon les trois principes que nous avons dégagés constitue certes une rupture dans notre logique habituelle de la délibération et de l'action, mais observer cette rupture requiert de dégager la continuité originelle sur le fond de laquelle elle survient. Ainsi il semblerait que Martin-Löf, dont la réflexion philosophique est imprégnée de Kant, soit tributaire du même « angle mort » que nous avions relevé chez ce dernier[96].

Pourtant Martin-Löf n'insiste-t-il pas sur le fait que les actes dénotés par les règles constructives des types sont bien celles que doit suivre l'investigation pour résoudre le problème ? Ces actes ne sont-ils pas conçus comme *productifs* ou *constructifs*, de la même manière que la délibération construit des procédures ?

Il est intéressant ici de suivre le logicien Sundholm, proche de Martin-Löf, lorsqu'il établit la distinction, que lui aurait indiqué Martin-Löf, entre la preuve comme objet construit, l'acte de la preuve, et les traces de cet acte. L'objet-preuve (*proof-object*) est ce qui est construit, le théorème qui contient sa propre démons-

96. Voir plus haut § 142.

tration, par l'acte mental du mathématicien qui réalise cette construction. Mais les traces que laisse le mathématicien – la preuve écrite, ou les diagrammes au tableau, donnent plutôt des indications permettant à d'autres mathématiciens de reproduire cet acte, qui peuvent être elliptiques ou au contraire donner des explications complémentaires. Sundholm cite une analogie qu'aurait donnée Martin-Löf : si on considère une randonnée à ski, le lieu d'arrivée de la randonnée est l'objet, la randonnée elle-même est l'acte de la preuve, et les traces des skis, ou des balises que l'on aurait laissé sur le chemin sont des traces utiles pour les randonneurs suivants[97].

Cependant, ces traces sont celles d'un randonneur *qui connaît déjà le chemin*, car Sundholm ne fait nulle mention de fausses pistes, d'hésitations, de demi-tours qu'il aurait dû soigneusement effacer par la suite afin de déterminer un chemin optimal pour les skieurs suivants, et leur éviter ses propres errances.

Dans un autre article, Sundholm parle, concernant les traces d'une démonstration mathématique, d'une publication dans un journal académique, mais également de traces de craie sur un tableau, de gobelets de café usagés, de notes gribouillées[98]. On espère qu'ici Sundholm évoque la trace d'une *recherche*, de longues heures passées à retourner le problème dans tous les sens, mais il n'en est rien. Cet exemple n'est pas commenté et il ne fait que s'ajouter à d'autres exemples, comme la publication académique, qui n'évoquent en rien l'hésitation de la recherche. Ces traces sont celles laissées par un mathématicien en pleine possession de ses facultés, qui déroule sans hésiter et sans faux-pas le protocole contenu dans l'objet-preuve.

Il ne faut pas chercher dans ces textes autre chose que ce qu'y vise Sundholm : l'analyse de la notion de preuve, afin de la distinguer aussi bien d'une interprétation subjectiviste que formaliste. Son intention n'est pas de réfléchir aux processus de la connaissance et de l'élaboration des preuves, ni de caractériser la créativité du mathématicien, comme il le dit explicitement[99]. Cependant, il nous semble significatif que, dans toutes ces analyses minutieuses, il ne soit fait nulle place à l'acte, non pas d'*exécuter* la preuve (ou de la trouver par une intuition quasi-somnanbulique), mais de la chercher et de la construire patiemment, avec de multiples allers et retours.

Sundholm et Martin-Löf méditent dans le cadre du « mathématicien idéal » de

97. (Sundholm 1993, p. 61-62).
98. (Sundholm 1998, p. 32).
99. (Sundholm 1993, p. 68).

Brouwer, mais le problème avec ce cadre est qu'on ne comprend pas très bien *qui*, finalement, effectue la preuve-objet, cette méthode idéale que construit l'enquête mathématique. Ce paradoxe a été relevé par Maronne (2017), que les mathématiciens indiquent en général des actes possibles, sans jamais donner l'air de les réaliser vraiment, puisque ceci « ne sert point pour cultiver ou récréer l'esprit, mais seulement pour exercer la patience de quelque calculateur laborieux[100] ».

§ 164. Mäenpää et l'art de l'analyse

La thèse de Mäenpää (1993), *the Art of Analysis. Logic and History of Problem-Solving*, est beaucoup plus proche de notre souci de faire place à l'investigation. En se fondant sur la théorie des types, Mäenpää cherche à reconstruire la logique des processus de pensée qui permettent de prouver des théorèmes ou de calculer des solutions d'un problème. Il analyse ainsi soigneusement les raisonnements de géomètres grecs, de Viète, Descartes, Newton et Galilée, en ne se confinant donc pas au seul domaine des mathématiques. Sa thèse principale est que l'antique distinction de la méthode en analyse et synthèse permet de penser les deux mouvements essentiels du raisonnement. La synthèse correspond à l'acte idéal évoqué par Sundholm et Martin-Löf ainsi qu'à l'objet-preuve qu'il construit. Mais celle-ci n'est possible que sur la base d'une analyse préliminaire, que Mäenpää propose de réinterpréter de manière *configurationnelle* plutôt que *directionnelle*, des termes qu'il emprunte aux travaux de Hintikka et Remes.

L'interprétation directionnelle de l'analyse, classique, la présente comme une investigation *à rebours* du théorème à prouver ou de la construction à réaliser, partant de ce qui est demandé pour en analyser les conditions, jusqu'à parvenir à des éléments connus ou disponibles, ou à une impossibilité. Pour Mäenpää, cette formulation peut certes convenir aux raisonnements logiques les plus simples, mais l'analyse moderne, telle qu'elle est inaugurée par Descartes grâce à l'algèbre, consiste plutôt à identifier la relation fonctionnelle qui connecte les inconnues aux données, c'est-à-dire la structure sous-jacente du problème à résoudre. C'est ainsi, par exemple, qu'il transforme les problèmes géométriques en systèmes d'équations algébriques dont chacune décrit une figure. L'approche fonctionnelle de la théorie des types permet, selon Mäenpää, de généraliser cette compréhension configurationnelle de l'analyse à tout type de problème, y compris ceux de preuves logiques.

100. Lettre de Descartes à Elisabeth, novembre 1643, citée par Maronne (2017).

Bien plus, l'axiomatisation de la géométrie euclidienne qu'il propose, proche de celle de Von Plato évoquée plus haut, doit permettre, selon lui, de refléter les raisonnements analytiques des géomètres grecs *dans leurs propres termes*, c'est-à-dire sans d'abord les transformer, comme le fait Descartes, dans un langage algébrique.

L'étude de Mäenpää nous semble très importante pour deux raisons :

1. D'une part, elle reconnaît qu'il y a *deux* raisonnements dans toute investigation scientifique. Le premier est l'investigation proprement dite, qui cherche une solution. Le résultat de cette investigation n'est pas seulement d'exhiber cette solution, mais également d'indiquer une manière directe de l'obtenir, sans qu'il y ait à réitérer sa propre démarche. Pour reprendre la métaphore de Martin-Löf en la déplaçant, le chemin qu'effectue le guide de montagne *n'est pas* en général celui qu'il va baliser pour les skieurs suivants. Que ces deux investigations correspondent aux méthodes que désignaient les Anciens par analyse et synthèse est une autre question, sur laquelle nous revenons sitôt.
2. D'autre part, elle explicite l'idée sous-jacente à la théorie des types, qu'il existe un formalisme universel pour la résolution de problème, qui étend les méthodes algébriques à n'importe quel domaine qu'il s'agisse d'analyse numérique, de géométrie, de logique, d'informatique[101]. Cette forme est celle d'une procédure fonctionnelle, qui transforme un contexte initial en un contexte final, et sa nature computationnelle est évidente.

Cependant ces résultats doivent être nuancés. Cette forme universelle s'applique d'abord au *second* raisonnement que nous évoquions à l'instant, celui qui est produit par l'investigation, c'est-à-dire à l'objet-preuve, ou encore à la synthèse, dans le vocabulaire de Mäenpää. Mais la thèse de ce dernier ne porte pas sur cela, qui lui semble évident : il ambitionne de dégager une telle forme pour l'investigation proprement dite, le *premier* raisonnement. Le titre de son ouvrage, *l'Art de l'Analyse*, évoquant *l'Art de Penser* de la logique de Port-Royal, suggère une portée prescriptive, que la première ligne du texte confirme :

> Cette étude développe une méthode générale de la résolution de problème, sur la base de la méthode analytique conçue par les géomètres grecs antiques[102].

Or l'investigation, telle qu'on peut l'observer empiriquement, ne se réduit pas en général aux raisonnements analytiques que Mäenpää trouve dans les écrits de Descartes ou Newton. D'abord, ces raisonnements sont restreints à des sujets scienti-

101. (Mäenpää 1993, p. 2-3).
102. (ibid., page liminaire).

fiques, tandis que nous nous intéressons également aux investigations techniques et pratiques en général.

Par ailleurs, ces raisonnements analytiques marquent la phase terminale de la recherche et présentent déjà un degré d'exactitude et de rigueur que l'investigation n'a pas en général. Ils avancent notamment de manière linéaire et systématique. Il est par exemple très important pour Mäenpää que la démarche analytique conserve à tout moment la *portée* du théorème à prouver ou de la construction à réaliser, c'est-à-dire qu'elle ne restreigne pas la spécification à des cas particuliers, afin de pouvoir être finalement convertie de manière légitime en synthèse. Or une telle exigence *n'est* en général *pas* recommandée à une recherche qui débute. Celle-ci est d'abord confrontée à des questions beaucoup plus élémentaires, qui concernent la formalisation du problème lui-même, les dimensions de description du contexte, la compréhension et la négociation des objectifs, etc., comme nous l'avons vu en première partie dans le cas spécifique de la programmation informatique. Une investigation qui débute est invitée à raisonner par cas particuliers, ou par simplifications, ou par analogies[103]. Elle doit construire le cadre de la réflexion, ou encore l'espace des possibles. L'analyse, telle que la décrit Mäenpää est par contraste une technique de « fin de partie », comme on dit au jeu de go ou aux échecs : elle permet d'assurer qu'un problème déjà suffisamment circonscrit est correctement réduit en ses derniers éléments.

Il est utile ici de se rappeler l'exemple de l'échiquier mutilé de Herbert Simon : la complexité de certains problèmes peut être radicalement réduite par un simple changement de perspective, ici de colorier les cases de l'échiquier. Ce « coup » ne peut être décrit comme une simple construction auxiliaire dans l'espace de problème initial, mais comme une réorganisation complète de celui-ci. Mäenpää montre certes comment l'analyse peut *contribuer* à la formation de concepts théoriques, mais il reconnaît que cette investigation contient d'autres ingrédients, comme la formation d'hypothèses, ou ce qu'il appelle l'induction exemplaire (*instance induction*) qui est le choix de donner une portée universelle à une règle dégagée de l'analyse d'un exemple donné.

L'investigation ne doit donc pas être définie, de manière étroite, comme résolution d'un problème bien posé, comme la seule invention d'une procédure ou *savoir-comment-faire*, selon le terme employé par Martin-Löf dans le texte cité plus haut. Elle comprend également la clarification de la spécification, du *savoir-quoi-faire*,

103. (Pólya [1945] 1990).

ce qui revient à *poser le problème*, à définir l'espace des possibles, comme nous l'avons affirmé avec force en première partie à propos de la programmation informatique.

Certes, il est toujours possible que des ratiocinations, c'est-à-dire des raisonnements qui se contentent de suivre pas à pas des procédures déjà établies[104], viennent s'intercaler au sein de de l'investigation afin de l'abréger ou de la rendre plus sûre, et il s'agit d'ailleurs là de leur fonction centrale. Dans les cas simples, nous l'avons dit, la procédure se substitue entièrement à l'investigation – le guide de montagne n'a plus à accompagner les skieurs une fois le chemin balisé. Mais dans d'autres cas, les investigations complexes qu'étudie Mäenpää, la métaphore adéquate nous semblerait être celle d'une ascension ardue de haute montagne, dont seules quelques étapes auraient pu être balisées – toutes les autres étant laissées à la charge de l'alpiniste qui devrait les inventer en fonction des conditions météo, des avalanches, des cascades de glace, etc.

Donc, lorsqu'on décrit de manière procédurale, comme le fait Mäenpää, une investigation, ou une partie de l'investigation, on ne décrit pas l'investigation elle-même, mais déjà son résultat, c'est-à-dire une procédure. Elle est déjà reconstruite idéalement, comme la conduirait le mathématicien idéal de Brouwer qui seul intéresse Martin-Löf, Sundholm, et finalement Mäenpää lui-même. Tout ce que cela signifie, c'est qu'il est possible d'établir des procédures pour construire des procédures, mais cela ne signifie pas que toute investigation n'est que l'exécution d'une procédure, car on retomberait là dans une thèse computationnelle de l'esprit[105].

§ 165. La valeur de l'axiomatisation

Il est utile, à ce stade, de revenir à la perspective défendue par Reuben Hersh, par laquelle nous avions ouvert ce chapitre. Comme toute axiomatique, le problème de la théorie des types serait qu'elle théoriserait *après coup* les travaux réels d'investigation des mathématiciens, manquant par là leur teneur essentielle :

> Parfois la solution [d'un problème] est un ensemble d'axiomes ! Je m'explique. Lorsqu'un fragment de mathématiques devient gros et compliqué, nous pouvons vouloir le systématiser et l'organiser, pour l'esthétique et pour la commodité. La façon dont cela se fait est de l'axiomatiser. Ainsi, un nouveau type de problème (ou « méta-problème ») surgit : « Étant donné un sujet mathématique spécifique, trouver

104. Sur notre emploi du terme *ratiocination* voir plus loin, § 173.
105. Voir plus haut, § 9.

> un ensemble attrayant d'axiomes à partir desquels les faits du sujet peuvent être facilement dérivés. » Tout ensemble d'axiomes proposé est une solution proposée à ce problème. La solution ne sera pas unique. Il y a une histoire des ré-axiomatisations de la géométrie euclidienne, de Hilbert à Veblen en passant par Birkhoff (le père). Dans le développement et la compréhension d'un sujet, les axiomes arrivent tardivement. Ensuite, dans les présentations formelles, ils arrivent en premier[106].

Dans le courant des années 1980, les partisans d'une philosophie attentive aux pratiques effectives des mathématiciens[107] ont ainsi critiqué sa réduction à la question des fondements des mathématiques, et rejeté dos à dos les différentes écoles fondationnelles (formalisme, logicisme, intuitionnisme) élaborées dans la première moitié du XX^e^ siècle, comme excessivement obsédées par l'axiomatisation théorique – faisant remonter ce travers à Euclide lui-même. Lakatos écrit ainsi :

> Un système [déductif] est euclidien s'il est la clôture *déductive* de ses énoncés de base qui sont supposés être vrais. […] Dans une théorie euclidienne, les énoncés de base vrais, situés au « sommet » du système déductif (généralement appelés « axiomes ») *prouvent*, pour ainsi dire, le reste du système […]. Le flux caractéristique [de la vérité] est sa transmission « vers le bas » depuis les axiomes jusqu'au reste du système — la logique est ici un organon de preuve[108].

On peut juger caricaturale cette description des entreprises d'axiomatisation, qui semble les réduire à leur squelette logique, mais elle doit être comprise comme une réaction contre un état d'esprit régnant à cette époque, qu'il s'agissait de renverser. Car pour Lakatos, cette focalisation des philosophes sur les problèmes d'axiomatisation et de fondation prend son origine dans une conception de la vérité mathématique comme certitude fondée sur l'apodicité des preuves et donc, ultimement, sur les axiomes. Dans ce schéma de pensée, il est compréhensible que la fondation des systèmes axiomatiques soit conçue comme primordiale. En écho à Lakatos et Hersh, Wang[109] dénonce l'axiomatisation théorique comme une activité tardive de compilation et d'organisation du savoir guidée par des « besoins esthétiques » et une conception « bibliothécaire » des mathématiques. On retrouve bien ici la critique du savoir théorique comme contemplation passive de vérités rendues transparentes par l'enchaînement des preuves.

Le projet constructiviste a une ambition fondationnelle, et il est donc visé en

106. (Hersh 1997, p. 6).
107. Voir plus haut, § 145.
108. (Lakatos [1986] 1998, p. 33).
109. (H. Wang [1961] 1998, p. 133).

partie par les critiques des philosophes des pratiques mathématiques. Knorr associait d'ailleurs Zeuthen à l'intuitionnisme, et critiquait les projets de reconstruction de la géométrie :

> Si l'on souscrit à une telle vision intuitionniste du domaine antique, il faut aussi reconnaître que les Anciens avaient une compréhension incomplète des exigences de leur position[110].

Chez les constructivistes en effet, toute preuve pourrait être décrite comme une preuve d'existence, puisqu'il s'agit d'exhiber un objet qui répond à une spécification, ce qui ne s'accorde certainement pas à la manière dont elles sont présentées par les géomètres antiques.

Afin de se représenter ce que pourrait être une théorie mathématique conçue comme un pur champ de problèmes, ainsi que Hersh nous y invite, on peut tenter d'imaginer une mathématique qui décrirait des actes concrets en se passant, à la limite, de signes écrits. Les études ethno-mathématiques[111] ont révélé des exemples d'activités constructives à caractère algorithmique, comme les jeux de ficelle[112] ou le tissage dans les sociétés andines[113], qui se déroulent dans des champs de règles strictes, et exhibent des résolutions de problèmes d'une complexité croissante. Ce type d'activités exhiberait la forme essentielle des mathématiques, comme activités certes adossées à des objets et à des pratiques concrètes, et néanmoins pures, en ce qu'elles se donnent à elles-mêmes leurs problèmes et leurs règles, dans la continuité d'une tradition.

Selon cette perspective pragmatique, la démarche de systématisation qui s'exprime pour la première fois dans les *Éléments* devrait être vue dans sa continuité avec ces pratiques. La géométrie, sans doute issue de domaines techniques divers comme l'architecture, l'arpentage, la menuiserie, etc. visait à rendre évidentes les relations qui existent entre des classes de problèmes dont on reconnut progressivement la similarité. Selon Sidoli (2018), l'entreprise d'Euclide, dans sa partie constructive tout au moins, est de décrire des procédures permettant de construire progressivement, de manière modulaire, des objets de plus en plus complexes, à partir d'une économie maximale de moyens. En d'autres termes, la systématisation renvoie ici à une démarche réflexive qui épure aussi bien les problèmes que leurs solutions procédurales, et qui les reformule de manière à montrer leur inter-

110. (Knorr 1983, p. 138).
111. Voir (Vandendriessche et Petit 2017) pour une introduction à ce champ disciplinaire.
112. (Vandendriessche 2016).
113. (Desrosiers 2012).

dépendance : quelle est, par exemple, la méthode la plus simple pour construire un angle droit, ou deux parallèles, ou le centre de gravité d'un triangle, pour bissecter ou trissecter un angle, pour doubler le volume d'un cube, etc. Tous ces problèmes, on le sait, ont des solutions qui dépendent les unes des autres, ou qui utilisent des procédures élémentaires communes. Dans la démarche théorique ou systématique qui est exprimée par les *Éléments*, chaque problème n'est pas énoncé séparément, mais avec tout son contexte. Le contexte d'un problème n'est pas seulement constitué par ses données initiales, mais également par le but qu'il s'agit d'atteindre et par les moyens qui sont autorisés, comme la règle et le compas dans les *Éléments*. Bien plus, le contexte implicite d'un problème, nous apprend Sidoli, contient chez Euclide tous les problèmes déjà résolus et théorèmes déjà établis auparavant, ce qui permet de se représenter le contexte des *Éléments* comme dynamique, s'élargissant au fil de la lecture, chaque preuve ou procédure venant s'ajouter au corpus des connaissances du géomètre comme disponible pour le problème suivant.

La théorisation d'un ensemble disparate de savoirs pratiques apparentés prendrait donc son origine dans le projet de leur ordonnancement dans un cadre commun. Du point de vue de son utilité, ce cadre garantirait d'abord de donner les *meilleures* solutions possibles aux problèmes couverts, notamment les plus économes (les solutions les plus courtes sont proposées), la factorisation des problèmes étant ici centrale (les étapes communes à plusieurs solutions sont identifiées comme solution d'un problème plus élémentaire). Par ailleurs, un cadre théorique n'indique pas seulement les plus courts chemins, il indique également les impasses (les problèmes impossibles à résoudre), les routes encore à défricher et celles qui seraient communes (les problèmes ouverts et interdépendants), etc.

Peut-on cependant se contenter de légitimer la démarche de théorisation par les « vertus épistémiques » qu'elle confèrerait au savoir ? L'ensemble de nos réflexions, au cours des trois derniers chapitres, indique le contraire. Si tel était le cas en effet, nous nous serions contentés des arborescences ramistes ou des combinaisons lulliennes. La systématicité ne vise pas d'abord à organiser le savoir mais à contrôler la variabilité des conditions d'application des règles, et cette seconde visée est ce qui rend possible la première. Comme nous l'avons vu, c'est parce que les opérateurs de construction de la géométrie sont inductifs qu'il est possible d'organiser son champ de problèmes de manière progressive, en allant des plus simples aux plus complexes.

Ainsi, les deux interprétations des mathématiques que nous avons analysées

dans ce chapitre pèchent chacune d'un côté. La perspective constructive nous permet de définir de manière exacte les principes de la « science des opérations », mais nous laisse démunis quand il s'agit d'expliquer comment celle-ci *se fait* – ce qui nous empêche de comprendre la nature commune de la délibération pratique et de l'investigation systématique, ainsi que la rupture qui s'opère entre elles. La perspective pragmatique au contraire, lorsqu'elle insiste sur la continuité entre les mathématiques et les autres pratiques humaines, va trop loin dans sa critique de l'axiomatisation et de sa valeur propre, en la réduisant à une activité de « bibliothécaire ». Cela ne nous aide pas à comprendre le rapport spécifique qu'a la programmation à la systématicité.

Il nous faut donc à présent réconcilier ces deux perspectives et chercher à comprendre comment la systématicité, dont l'esprit théorique relève, est une *fonction* de l'investigation, sinon « naturelle », du moins qui relève de sa possibilité interne.

Ce qu'il nous faut donc à présent élaborer, c'est un cadre où toutes ces notions de *problème, investigation, procédure, théorie, machine* et *programmation* seraient articulées les unes aux autres, et où la théorie des types de Martin-Löf, comme logique des problèmes, pourrait devenir une théorie de l'investigation.

Chapitre 10

La théorie de l'investigation (Dewey)

§ 166. Introduction

Le chapitre précédent a permis de répondre, dans une certaine mesure, aux deux questions fixées à cette partie : à celle qui demandait comment concevoir la nature commune des règles du jugement et de l'action, ainsi qu'à celle qui demandait ce qu'était, au fond, la systématicité dont les mathématiques avaient montré la possibilité.

Si en effet l'activité de ce que Kant appelle *entendement* est essentiellement *résolution de problème*, au sens le plus large qu'on pourrait donner à cette expression : savoir quoi juger d'une situation, et savoir quoi y faire (des fins étant données), alors la théorie des types donne la forme logique unifiée et générique que peuvent prendre de tels *savoirs des règles*. Les règles d'un concept sont, d'une part, les différentes manières d'accéder à ses objets, sous certaines conditions, et d'autre part, celles par lesquelles ils peuvent eux-même être conditions d'autres objets. On peut donc, dans un cadre théorique dont les concepts élémentaires ont été définis, exprimer toute forme de problème complexe comme une question d'accessibilité d'un objet complexe, à l'aide d'une composition d'actes élémentaires. On voit dans ce cadre qu'il n'y a pas de distinction essentielle à faire entre des théories du savoir et des théories du savoir-faire, ainsi que le soutenait déjà Kant. Comme le dit Martin-Löf, il est indifférent *à la forme* de la théorie que les objets traités soient des équations ou des gâteaux, et que leurs actes de construction soient des inférences ou des gestes de cuisine[1]. En étudiant la géométrie, nous avons certes vu que toute pratique humaine complexe impliquait de traiter de nombreux types d'objets dif-

1. Voir plus haut, p. 635.

férents, et que des actes « intellectuels » pouvaient souvent servir de validation ou de spécification à des actes « techniques » ; mais il ne s'agit là que d'une division du travail relative à un domaine pratique donné, et non d'une division logique ou épistémologique concernant les types de savoir humain – puisqu'au contraire leur interdépendance était requise pour qu'une pratique maîtrisée puisse advenir.

La systématicité d'un domaine est acquise lorsque les règles élémentaires de cette interdépendance sont élucidées, de sorte que tout problème puisse être explicité à l'aide de ces règles. On peut caractériser un domaine comme fini selon le nombre de problèmes qui peuvent être générés à l'aide de ces règles. C'est le cas, on se le rappelle, de la syllogistique aristotélicienne, dont le nombre de théorèmes dépend du nombre d'axiomes, ou de la diérèse platonicienne, dont les règles sont les genres et les différences spécifiques. La systématicité devient intéressante lorsque le domaine à élucider est infini. Ce que montre la théorie des types, c'est qu'il faut alors découpler l'espace de problème lui-même, qui peut être infini, des règles de formulation et de résolution des problèmes, qui sont finies et qui doivent accommoder d'une manière ou d'une autre trois principes : un principe de localité (car seule une portion finie de l'espace peut être considéré par une règle donnée), un principe de composition (par lequel un problème résolu peut former la base d'un nouveau problème), et un principe d'induction qui donne le schéma élémentaire par lequel cet espace peut être parcouru exhaustivement, aussi bien pour définir de nouvelles règles que pour les appliquer. Dans le cas en effet des problèmes ayant une variété finie de conditions, la résolution de problème peut se contenter de les énumérer toutes ; mais dans le cas d'une variété infinie, une règle permettant de les parcourir exhaustivement doit d'abord être donnée, et c'est relativement à cette règle qu'une solution systématique doit être exprimée, c'est-à-dire définie de manière inductive. Comme nous l'avons dit, la définition récursive des entiers naturels est la forme la plus simple d'une systématicité infinie.

Cependant, au terme de cette réflexion, nous nous sommes demandé si une telle fixation, dans un langage formel comme la théorie des types, de la logique de la résolution de problème, n'amenait pas à oublier à nouveau l'essentiel, c'est-à-dire l'attitude délibérative qui la commande. La théorie constructive des types semble bien décrire la logique de la résolution de problèmes, mais seulement si on entend par problème celui qui a été *reconstruit* au cours de l'investigation, et par « résolution » le fait accompli qui présente sa justification et non le processus même qui l'a patiemment construite, au terme de nombreuses erreurs et hésitations. Les

travaux de Mäenpää pointent vers cette distinction nécessaire, mais de manière insuffisante selon nous ; elle ne saurait être identifiée à l'antique opposition entre analyse et synthèse.

Il nous faut donc quitter le domaine de la logique et du calcul formel, et observer la délibération « en train de se faire », les problèmes tels qu'ils se présentent à elle, et les théories en tant qu'elles sont mobilisées pour les résoudre. C'est seulement une fois acquis cet ancrage dans l'*expérience de la connaissance* qu'il sera possible d'y trouver une place adéquate pour la théorie des types, et pour la logique des problèmes qu'elle propose ; c'est seulement là que nous pourrons comprendre ce qu'est la résolution *systématique* de problème, relativement à ses manières simplement *pratique* ou *technique*.

Notre questionnement porte donc à présent sur l'*activité* même de la délibération, c'est-à-dire sur la pensée réflexive qui est à l'œuvre dans la résolution de problème. Ce qui caractérise toute délibération difficile, ce sont ses hésitations, ses errements, ses reconceptualisations réussies ou avortées. Doit-on, à l'observation de ces mouvements en apparence désordonnés, conclure qu'elle procède par simple association d'idées, et qu'elle ne peut donc qu'être l'objet d'études psychologiques, ainsi que le voulaient les positivistes logiques ? Ou doit-on supposer, avec les premiers cognitivistes comme Herbert Simon, qu'elle exécute des algorithmes élémentaires dont la complexité du comportement résulte simplement de celle de son environnement externe et interne (sa mémoire) ? Le problème de ces deux solutions est de ne pas faire droit à la caractéristique même de l'investigation que nous voulons cerner, à savoir qu'elle est une recherche et une tension vers…, ou encore une intention, selon les termes d'Anscombe. Elles adoptent d'emblée une perspective externe qui la décrit comme un processus naturel quelconque, ou comme un mécanisme.

L'intérêt de la perspective de Dewey est de poser que cette tension vers la résolution de situations problématiques est un fait élémentaire de la vie, à la fois naturel et intentionnel. De manière significative, il place le point de départ de toute pensée réflexive dans la rencontre d'un problème, terme auquel il propose de donner une portée maximale :

> Si nous acceptons d'étendre la signification du mot *problème* à tout ce qui – aussi léger et banal soit-il – rend perplexe et défie l'esprit de manière à rendre la croyance incertaine, [alors nous pouvons dire qu']il y a un véritable problème ou question

impliquée dans cette expérience de changement soudain[2].

La pensée réfléchie peut donc être définie comme une activité de résolution de problème, ou encore de restauration d'un équilibre dans la croyance[3]. Sa *Logique* de 1938, une œuvre tardive qui systématise ses réflexions sur la nature de la connaissance développées très tôt, définit ainsi la logique comme une *théorie de l'investigation* (*a Theory of Inquiry*)[4].

Ce chapitre reprend donc à neuf la première question de cette partie[5], en l'adressant à Dewey : comment concevoir l'identité des règles du jugement et de la *transformation* du jugement dans l'investigation ? et en quelle mesure cela nous permet-il de conclure à la nature commune des règles du jugement et de l'action, comme l'affirme Kant ? Nous estimons adéquatement résolue notre seconde question directrice, concernant la systématicité mathématique, et n'y revenons pas.

La section 10.1 développe la thèse fondamentale de Dewey que nous venons d'expliciter, et montre l'unité de la notion de *situation problématique* à travers tout le spectre de l'expérience humaine : Si l'investigation est le développement d'un fait élémentaire de la vie – la recherche d'un équilibre avec son milieu, alors tous les problèmes, des plus simples aux plus subtils théoriquement, peuvent être vu comme l'explicitation discursive de cette tension élémentaire qui doit être d'abord ressentie avant de pouvoir être posée dans le langage. Certaines de ces tensions sont causées par la résolution d'autres tensions, et donc par la discursivité elle-même, comme, par exemple, la difficulté que nous avons parfois à mettre en cohérence deux formulations d'une même situation. Cependant toutes ces tensions, qui se *composent* entre elles et s'*infèrent* les unes des autres, s'ancrent de manière ultime dans un même et unique rapport de la personne humaine à son environnement, qu'il faut ainsi concevoir comme dynamique. Cette composition et cette inférence des problèmes entre eux ne peut apparaître que chez les êtres dotés de langage. Le langage permet en effet à l'individu de *marquer* ses habitudes et ses tendances

2. (Dewey [1910] 1997, p. 9).
3. Voir sur ce point (Madelrieux 2016, p. 97).
4. Nous citerons cet ouvrage par l'abréviation *LTI* (*Logic, a Theory of Inquiry*), suivi du numéro de page dans son édition des œuvres complètes de Dewey par Ann Jo Boydston. Nous suivons l'usage de désigner les volumes de cette édition par les abréviations désignant ses trois séries, EW (*Early Works*), MW (*Middle Works*), LW (*Later Works*), suivi du numéro de volume, puis de la page. La *Logique* correspond au volume LW 12.
5. Voir plus haut, § 144.

naturelles à agir comme des objets et de les considérer dans des plans symboliques, où il peut suivre leurs conséquences à loisir.

L'investigation est donc une forme d'action silencieuse qui peut se permettre des explorations et des errements, *car justement elle ne prête pas à conséquence*. Le jugement est l'achèvement de l'investigation, en ce qu'il est l'acte par lequel l'agent se décide à agir – à *appliquer* son plan symbolique à la situation vécue. C'est cette décision qui est le modèle du jugement logique traditionnel et non l'inverse. Ce n'est donc pas parce qu'elle est imparfaite que la délibération procède par hésitations et erreurs, et reprise itérative du problème. Elle est *tout entière* un mouvement de reconstruction, dans un réseau de significations donné, de la situation problématique jusqu'à ce que la solution – la décision d'agir – apparaisse « évidente ». La formulation *a posteriori* de cette délibération, lorsqu'elle est achevée, dans la procédure ou la preuve, *efface donc essentiellement ses traces* qui ne lui apparaissent plus, rétrospectivement, que comme des erreurs de jugement inutiles à la solution. La procédure n'est donc *pas* un acte de l'expérience, comme la délibération, mais un acte *idéal* proposé aux examinateurs externes du jugement – soit qu'ils veuillent s'assurer de sa légitimité, soit qu'ils veuillent pouvoir le reproduire ou s'en inspirer dans le futur (section 10.2). Les mathématiques sont ainsi, essentiellement, des moyens procéduraux qui permettent le contrôle et la formulation systématique des reconstructions idéales du jugement – ce qui nous permet d'avancer la thèse que la théorie constructive des types est un développement et une formulation exacte des intuitions logiques de Dewey (section 10.3).

Les théories scientifiques sont imprégnées de mathématiques parce qu'elles ne sont rien d'autre que des *systèmes de contrôle de l'inférence*, c'est-à-dire le résultat d'investigations de second degré qui ne cherchent pas à résoudre des problèmes particuliers, mais à établir une fois pour toutes les règles par lesquelles peuvent être exprimés et résolus tous les problèmes d'un domaine donné (section 10.4). De la même manière nous avancerons que les systèmes institutionnels et techniques sont également des constructions de second degré, qui ne visent pas à satisfaire des objectifs immédiats, mais à contrôler et à assurer la réussite de tout un champ pratique donné (section 10.5).

En conclusion, nous proposons une synthèse des trois parties centrales de ce travail, à partir de la théorie de l'investigation de Dewey, qui lui sert de creuset (section 10.6). Cette synthèse est le socle sur lequel nous allons, lors de notre dernière partie, répondre à notre question centrale, concernant la nature et l'ef-

fectivité de la programmation comme résolution systématique de problème, ainsi qu'au rôle nécessaire qu'y joue la machine.

Ce chapitre peut, à bien des égards, être lu comme un exposé didactique de la théorie de la connaissance de Dewey. La *Logique* de 1938, que nous suivons prioritairement, est un texte réputé d'un abord difficile et notre exposé reflète ainsi notre propre effort d'appropriation. Aussi la lectrice familière avec Dewey pourra-t-elle passer rapidement sur de nombreux développements, notamment dans les premières sections.

Cependant notre questionnement spécifique – qui vise la programmation et l'informatique – nous conduit à mettre en valeur des aspects moins connus de sa réflexion épistémologique, notamment concernant les mathématiques, ainsi que son insistance sur le caractère procédural de la connaissance et sur l'opposition massive qu'il propose entre moyens procéduraux et moyens matériels, qui a été très peu relevée jusqu'ici par le commentaire. Il importait en particulier pour nous de montrer la possibilité d'un rapprochement entre Dewey et les théories constructives, sans que ce geste apparaisse incongru, mais au contraire comme un prolongement naturel de pistes esquissées par Dewey.

En effet, notre problématique peut sembler de prime abord tout à fait étrangère aux centres d'intérêt de Dewey. Nous n'avons pas trouvé trace que le philosophe, pourtant resté très actif intellectuellement jusqu'à sa mort qu'en 1953, ait participé à la fascination collective de l'Amérique pour les « cerveaux électroniques » apparus après la seconde guerre mondiale, et qu'il ait perçu leur intérêt proprement conceptuel. Par ailleurs, comme le remarquent Burke, Hester et Talisse (2002, p. vii), la récusation par Dewey des développements contemporains de la logique formelle, qu'il attaque notamment à travers la figure de Russell, pouvait faire passer sa *Logique* comme anachronique dès sa parution en 1938, après une décennie faste pour la logique formelle, marquée par les travaux de Gödel, Tarski, Church et Turing.

Notre démarche est pourtant bien de tenter de prolonger les perspectives que Dewey propose sur la connaissance afin d'achever de construire le cadre épistémologique esquissé dans les chapitres précédents, autour entre autres des notions de *délibération* (ou *résolution de problème* ou encore *investigation* locale), de *règle*, et de *construction*. En particulier, il nous semble possible d'expliquer la possibilité et la puissance de la programmation des machines à partir des réflexions de Dewey sur la nature et la fonction de l'investigation rationnelle. Cela implique qu'il doit

être possible, dans le cadre de sa *Logique*, de rendre compte des formalismes sur lesquels se fonde l'informatique, comme la théorie des types de Martin-Löf examinée au chapitre précédent. Il nous importe de souligner que notre priorité est de construire un cadre épistémologique pertinent pour nos questions directrices, et non de transcrire littéralement à la pensée de Dewey, même si nous pensons rester fidèle à l'esprit de ses intuitions. Nous avons conscience de le tirer dans de nouvelles directions, mais il s'agit là d'un geste qu'il appelait lui-même de ses vœux[6].

Notre exposé est centré sur la *Logique* de 1938, qui constitue une vaste synthèse et un aboutissement de la philosophie de la connaissance de Dewey, mais nous ferons certains emprunts à d'autres textes. D'une manière générale, la pensée de Dewey présente une grande continuité et cohérence, malgré les inflexions et enrichissements successifs d'une réflexion logique commençant en 1890, que Johnston (2014, 2020) a mis en évidence[7].

Nous traduisons *inquiry* par *investigation*, contrairement à l'usage des traducteurs français qui utilisent en général *enquête*. Ce dernier terme a pour avantage de peut-être mieux refléter la portée très large que lui assigne Dewey, puisqu'il l'inscrit dans l'activité biologique de l'organisme – quel qu'il soit – qui cherche continûment à rétablir un équilibre avec son milieu, définissant l'*inquiry* comme « la transformation contrôlée ou dirigée d'une situation indéterminée en une autre, déterminée […] de manière à convertir les éléments de la situation originale en un tout unifié[8]. » Cependant, ces considérations très générales restent à l'arrière-plan de notre propos, qui s'intéresse uniquement à la résolution discursive de problèmes, et en particulier à sa formalisation procédurale. Le terme *investigation* est plus naturel dans ce cadre, se prêtant plus aisément à des associations avec les qualificatifs *mathématique* ou *scientifique*, que nous ferons couramment. L'emploi de ce terme assure également la continuité de notre réflexion avec la *ζήτησις* aristotélicienne, car nous tenons que Dewey traite bien de la même chose.

D'une manière générale, nous avons tenté d'éviter le vocabulaire technique de la *Logique* de Dewey dès qu'il n'était pas directement utile à notre propos, de

6. *LTI*, p. 5.
7. Les autres ouvrages de référence sur la *Logique* qui ont servi à notre synthèse sont celui de Burke ([1994] 1998) et l'ouvrage collectif édité par Burke, Hester et Talisse (2002). Nous avons plus généralement vérifié nos assertions dans les ouvrages sur Dewey de Hildebrand (2008), Hickman (1992) et de Madelrieux (2016).
8. *LTI*, p. 108.

manière à rendre notre lecture par la suite plus directement applicable à nos questions. Nous utilisons notamment *expérience* au lieu d'*existence*, et *empirique* au lieu d'*existentiel*. Dewey utilise ces termes afin d'insister sur le caractère *vécu* de l'expérience, qui est la réalité même, puisqu'il est pour lui illusoire de vouloir chercher un quelconque noumène derrière les phénomènes[9]. Une *signification existentielle* se réfère ainsi pour lui à une expérience potentielle, par exemple que le feu brûle. Il nous a semblé inutile d'introduire ici ces distinctions qui pourraient être source d'autres confusions. Pour la même raison, nous utilisons également *proposition empirique* au lieu de *proposition générique* ; les propositions génériques sont des propositions existentielles générales (comme « le feu brûle ») ; Dewey emploie ce terme pour les distinguer des propositions universelles, qui pour lui ont un contenu simplement logique.

10.1 Expérience et signification

Dans cette première section, nous montrons l'unité fondamentale de l'investigation à travers toutes ses formes, des plus simples aux plus élaborées : c'est qu'elle n'est elle-même que la forme discursive d'un phénomène élémentaire du vivant, celui par lequel l'agent tente de restaurer un équilibre avec son milieu. Cela nous conduit à exposer quelques notions fondamentales de la théorie de la connaissance de Dewey : l'unité de l'expérience vécue, qui précède logiquement ses parties (§ 167), l'action comme résolution d'une tension dans l'expérience (§ 168), l'inférence et la signification qui permettent l'action réfléchie (§ 169) et qui nécessitent pour ce faire le langage (§ 170). Nous exposons enfin les deux formes de propositions qui découlent de la nature du langage comme système de significations, à savoir les propositions empiriques et universelles (§ 171).

§ 167. L'unité de l'expérience

La notion de résolution de problèmes est si fondamentale pour Dewey qu'il l'inscrit dans l'activité élémentaire des organismes vivants tendus vers le rétablissement de leur équilibre avec leur milieu : faim, danger, désir sexuel, etc. Il n'y a pas alors, à proprement parler, *problème* puisqu'il n'y a pas conscience ou réflexion, mais une *situation problématique* qui se présente à l'individu comme un besoin,

9. Voir par exemple MW 6 : 66.

une gêne, une envie. L'intentionnalité, le *trying to get*... d'Anscombe évoqué plus haut (section 8.2) se trouve déjà là, dans ce phénomène élémentaire du vivant.

Dewey décrit la situation problématique comme une rupture dans l'*unité de l'expérience*. Cette dernière notion est fondamentale et a plusieurs dimensions. Elle dénote d'abord l'harmonie entre un individu et son milieu, le fait que l'activité (ou la passivité) du premier n'est pas troublée ou empêchée par le second – comme trouver aisément la nourriture là où on s'attend qu'elle soit, ou dormir sans être dérangé. Dans cet état de continuité la conduite de l'individu est tout entière *habitude*, ou activité adaptée à son milieu. L'habitude est ancrée dans le fond du fonctionnement organique de l'individu, dont les organes se comportent de manière réglée et ont une marge d'adaptation à la situation (respirer, saliver, dormir...). Ces activités organiques sont intégrées à des unités plus globales, réflexe coordonné, activités sensorimotrices, conduites apprises. La première continuité de l'expérience est donc spatio-temporelle. Il n'y a pas de sens à définir l'activité de manger comme une somme de mastications, ou comme le fonctionnement conjoint de la bouche, de l'estomac, du foie, etc. Ces distinctions sont faites *a posteriori* dans la description rationnelle de l'expérience du manger, mais celle-ci, lorsqu'elle n'est pas perturbée, est une et continue à travers le temps et les différents organes qui y contribuent. Par ailleurs, dans ces situations habituelles l'individu n'est pas conscient d'une séparation entre le milieu et lui, et son expérience n'est ni objective ni subjective : ces catégories de la discursivité n'ont pas d'application ici. Ainsi, l'expérience fondamentale n'est pas celle du feu, mais celle du « ça brûle », qui conjointement comprend la chaleur, l'âcreté, l'intensité lumineuse et mobile des flammes, phénomènes dont il n'y a pas de sens à se demander s'ils sont subjectifs ou objectifs. Cette continuité signifie qu'il n'est pas pertinent non plus d'opposer les expériences actives de l'organisme (comme marcher) et passives d'observation de changements de l'environnement (comme brûler). On devrait dire « ça marche » tout autant que « ça brûle » pour décrire les expériences fondamentales, infra-discursives, de l'individu, et Dewey les conçoit toutes sous la même catégorie de l'*activité*, sans considérer qu'il existe une différence fondamentale entre celles qu'on attribue à l'organisme et celles qu'on attribue au milieu naturel : toutes n'adviennent que dans l'interaction entre l'un et l'autre. C'est une abstraction de penser que « marcher » n'est pas tout autant une expérience *de* l'environnement qu'éprouver la chaleur du feu, et réciproquement, que se *tenir* face au feu n'est pas tout autant une activité que marcher.

Ce point est essentiel pour la compréhension de la logique de Dewey et il est donc utile de s'y attarder[10]. Il n'y a rien de tel qu'une *expérience purement objective du monde* où l'individu recevrait, dans une passivité totale, « sans rien y mettre du sien », des *données* provenant du réel situé face à lui. Dewey opère ainsi une déconstruction systématique des termes qui évoquent une telle idée, comme *observation, perception, sensation*. Il est aisé de reconnaître la dimension opérationnelle et intentionnelle de l'*observation*, qui est du même registre que le protocole expérimental. Observer un fait suggère une attention dirigée et motivée par l'investigation, un protocole, même sommaire, voire des instruments, ainsi qu'une utilisation possible du résultat dans la suite de la recherche[11]. Quant à la *perception*, si on entend par là la conscience générale du monde extérieur, elle est également active, en ce que la conscience est une attention dirigée, qui ne se met en éveil que de manière intermittente[12] – Dewey fait ici équivaloir perception avec *awareness*, qu'en français on pourrait traduire par *qui-vive*[13]. Et si l'on parle de perceptions sensibles élémentaires, comme le *« rouge – ici »* de Russell, Dewey affirme que la psychologie les étudie comme s'il s'agissait d'observations[14], mais en passant sous silence le contexte requis pour que celles-ci aient lieu. Remarquer du rouge dans le champ visuel est un événement plutôt exceptionnel, qui suppose un contexte inhabituel ou un intérêt particulier. Le « ici » dans *« rouge – ici »* pointe certainement vers des éléments moteurs actifs dans la perception[15]. Et si on veut se référer non pas même à la perception, mais à la sensation pure comme stimulus corporel périphérique, par opposition à l'attention centrale, il faut alors expliquer comment la distinguer des sensations de faim, de fatigue, des émotions, et surtout comment elle pourrait parvenir *pure* aux centres nerveux de l'organisme, sans se colorer de tous ces autres facteurs qui déterminent son état. Cet état ne peut être compris comme une somme de données passives convergeant vers un seul centre décisionnel qui renverrait aux organes du corps des instructions d'action, mais il est lui-même tout entier dans un état d'activité qui filtre et compose des milliers d'excitations internes et externes en fonction de sa disposition globale, elle-même

10. Voir à ce sujet (Madelrieux 2016, p. 123).
11. Par exemple (*LTI*, p. 116, 136).
12. LW 1, 233.
13. On trouve de nombreuses occurrences de cette équivalence dans LW 1, 226-235, par exemple.
14. *LTI*, p. 153.
15. LW 4, 172.

influencée par sa situation passée et par ses anticipations du futur[16].

§ 168. L'action

L'action est la réponse de l'individu à la situation problématique, sa tentative de la résoudre pour rétablir l'équilibre avec l'environnement. Les béhavioristes analysent l'action comme un conditionnement, et il y a en effet renforcement de l'habitude comme manière générique d'agir lorsque celle-ci est efficace dans une situation donnée. Mais le conditionnement est plutôt une conséquence de l'action que l'inverse, car il n'existe pas de situation « donnée », et sa perception même est déjà façonnée par un certain horizon d'attente : quels objets, ou couleurs, ou odeurs sont remarqués, comment ils sont associés les uns aux autres, tout cela dépend de la tension dans laquelle se trouve l'organisme. Ainsi, la *même* vision d'une proie excitera l'animal ou non, selon qu'il a faim. Il faut donc plutôt dire que c'est l'intention générique de s'adapter à son milieu, de répondre à ses différents besoins qui conduit l'animal, par tentatives successives, à reconnaître différentes situations pertinentes *et* leurs réponses adéquates, à les distinguer en situations de plus en plus particulières. On postule d'abord l'identité de l'expérience actuelle avec les expériences passées[17], et ce n'est que l'échec d'une action générique dans une situation insuffisamment analysée qui appelle à une action plus précise, mise en relation avec de nouveaux traits qui y sont reconnus. Le lion pourchasse les zèbres, mais plus spécifiquement ceux qui sont isolés, car il *perçoit* qu'ils sont plus vulnérables. L'action est d'emblée donnée comme une généralité qui s'affine progressivement.

La situation problématique et l'action qui y répond forment donc ensemble un « tout » qui est l'expérience d'une tension. Il ne faut pas voir les expériences complexes comme des compositions d'expériences unitaires que seraient les stimuli sensibles, car ceux-ci sont commandés par l'action globale qui répond à une situation globale. Toute expérience est également un réseau d'expériences potentielles, que l'action peut développer dans une direction ou une autre. Dewey illustre cette thèse en prenant l'exemple de la poursuite d'une proie :

> Le soi-disant stimulus, étant l'état total de l'organisme, se meut de lui-même, à cause des tensions contenues dans ces activités de poursuite qu'on appelle la réponse. […] Dans sa *relation* avec des activités spécialisées, le stimulus persiste tout au long de

16. LW 1, 251.
17. MW6, 381.

> la poursuite, bien que son contenu réel change à chaque étape. Lorsque l'animal court, les excitations sensorielles spécifiques, tactiles, olfactives et visuelles, se modifient avec chaque changement de position, de terrain, d'objets (buissons, rochers) qui apparaissent successivement ; et elles changent également d'intensité à chaque évolution de la distance avec la proie[18].

On voit également à cet exemple, qui montre une forme de continuité dans la tension du prédateur, qu'il ne faut pas opposer de manière binaire l'habitude, comme expérience de continuité de l'organisme avec son environnement, et l'action, comme expérience d'une tentative dirigée de rétablir cette continuité après une rupture. L'action doit toujours se déployer sur la base d'habitudes plus élémentaires de l'organisme, et d'autre part, sauf peut-être le sommeil profond, n'importe quel état de contentement doit affronter, à tout moment, des alertes, des excitations, des micro-ruptures dans l'expérience. Il y a, à tout moment, aussi bien séparation que continuité.

Par ailleurs toute action réussie réagit sur l'habitude. Elle est apprentissage, en ce qu'elle affine la relation de l'animal à son environnement : la coordination de l'animal s'améliore en réponse aux conditions d'environnement qu'il reconnaît, mais également ces conditions elles-mêmes sont modifiées, en ce qu'il apprend à mieux les reconnaître, à les rechercher, à les provoquer (par exemple par une olfaction active, ou une nouvelle posture du corps). Ces ajustements entre l'animal et son environnement créent à leur tour de nouvelles situations problématiques et ainsi de suite – ce processus n'est rien d'autre que le développement de l'animal – l'adaptation continuelle à son milieu[19].

L'unité de l'expérience signifie donc celle de sa durée, celle de l'individu et de son milieu, de l'activité et de la passivité, de l'intérieur et de l'extérieur ; elle est même unité de l'unité et de la rupture dans l'expérience, comme nous venons de le voir, et intégration de l'action dans l'habitude. Toutes ces catégories appartiennent à la discursivité qui n'est qu'une modalité ajoutée à l'expérience. Elles ne sont donc pas pertinentes pour la décrire. Aussi est-ce dans l'observation de l'animal que cette unité originelle de l'expérience peut le mieux se concevoir :

> Pour saisir les sources de l'expérience esthétique, il faut donc recourir à la vie animale qui a lieu en-deçà de l'échelle humaine. Les activités du renard, du chien et de la grive peuvent *a minima* être des rappels et des symboles de cette unité de l'expérience que nous fractionnons lorsque le travail est labeur et que la pensée nous retire

18. *LTI*, p. 36-37.
19. *LTI*, p. 41-42.

> du monde. L'animal vivant est pleinement présent, tout entier là, dans chacune de ses actions : dans ses regards méfiants, ses reniflements aigus, le dressement brusque de ses oreilles. Tous les sens sont également sur le qui-vive. En le regardant, vous voyez le mouvement se fondre dans la sensibilité et la sensibilité dans le mouvement – ce qui fait cette grâce animale avec laquelle il est si difficile pour l'homme de rivaliser. Ce que la créature vivante retient du passé et ce qu'elle attend de l'avenir opèrent comme des directions dans le présent. […] Le passé absorbé dans le présent continue sa course. Il presse vers l'avant[20].

§ 169. L'inférence et la signification

Comme on vient de le voir dans le texte immédiatement cité, on peut dire d'une certaine manière que, dans l'unité de l'expérience, l'animal est en attente (*expectation*), ou en anticipation de l'expérience à venir. Cependant, hormis ce texte, Dewey associe peu ces termes à la vie animale. Car une attente, à proprement parler, est la représentation d'une chose absente. Ainsi on ne dirait pas que le lion *s'attend à* ce que ses griffes déchirent sa proie. Une attente doit donc être marquée d'une manière ou d'une autre, fût-ce simplement *remarquée*. Cela se produit, par exemple lorsqu'à la vision d'une fumée, on s'attend à rencontrer le feu, si l'on va dans telle direction. Dewey parle ici de signe naturel.

Il y a *inférence* lorsque cette attente est formulée comme *devant avoir lieu d'une manière générale*. Le fondement (*ground*) de cette généralité dans l'attente réside nécessairement dans une opération : « Si je fais ceci, alors ceci se passera », car, comme on l'a vu plus haut c'est l'activité qui est essentiellement générique, c'est-à-dire l'habitude, qui réagit de manière générique à une situation qu'elle reconnaît de manière générique. Ainsi l'enfant infère que le feu brûle de l'attente qu'il se forge que, chaque fois qu'il approchera la main de la flamme, il éprouvera une brûlure. Dewey donne un autre exemple éclairant à la suite de celui-ci :

> Les Égyptiens attendaient les éclipses à des dates précises. Dans la mesure où les événements passés avaient été suffisamment analysés pour fournir le fondement de l'attente, celle-ci participait de la nature de l'inférence. Dans la mesure, cependant, où de simples événements temporels étaient le fondement de la prédiction, cette dernière n'était pas une inférence dans son sens logique définitif. Elle est devenue telle lorsqu'il fut établi que certains modes constants d'opérations naturelles (*certain constant modes of natural operation*) étaient la raison pour laquelle certaines

20. LW10,24

> conjonctions de conditions circonstancielles pouvaient être utilisées pour fonder une prédiction[21].

Dewey distingue ici deux niveaux d'inférence. Dans la première, on se fonde uniquement sur une certaine rythmicité temporelle des éclipses ; il y a inférence parce qu'on sait *compter* le temps, mais il ne s'agit pas là d'une « opération naturelle ». Cette dernière expression n'est pas très courante chez Dewey, mais on trouve dans un autre texte, chronologiquement proche, confirmation qu'elle signifie les changements naturels dans les choses « physiques », par opposition aux opérations biologiques qui sont ultimement fondées en eux[22]. Dans la seconde forme d'inférence, qui est celle à laquelle parvinrent les astronomes égyptiens, les « modes constants d'opération naturelle » désignent donc les mouvements des astres, dont une certaine configuration (la lune passant devant le soleil) permet de prédire une éclipse, grâce à des calculs[23].

Il semble cependant y avoir un tour de passe-passe dans le raisonnement de Dewey, car l'opération du soleil ne semble pouvoir être comparée que de manière métaphorique à celle de compter, ou à celle de l'enfant qui avance la main vers le feu. Or c'est là justement que nos remarques préliminaires sur la nature de l'expérience prennent leur importance. Le mouvement du soleil ne doit pas être dissocié de l'*acte de son observation* qui ne peut certainement pas se limiter à une simple perception, mais qui exige un protocole de mesure et de consignation du résultat permettant de le comparer à des actes passés. L'inférence de l'éclipse représente alors un dispositif opérationnel complet d'observation et de calcul, qui est l'astronomie égyptienne.

Avec l'inférence on se situe donc au seuil de la logique, c'est-à-dire du langage. Réaliser une inférence, affirmer le caractère général d'une attente, suppose en effet qu'on puisse *marquer* à volonté les expériences.

§ 170. Le langage

Cette compétence symbolique, le langage, est un fait fondamental de l'anthropologie de Dewey, et elle doit être rapportée à deux autres caractéristiques de la personne humaine : sa vie en société et sa compétence technique. Dewey entend le langage en un sens large, y incluant non seulement la parole et l'écrit, mais

21. *LTI*, p. 250.
22. LW 11, 108.
23. LW 8 : 199.

également le dessin, le rite, le monument, tout ce par quoi les personnes peuvent communiquer entre elles[24]. Il est donc corrélé à l'émergence d'une communauté culturelle, c'est-à-dire de la formation d'habitudes sociales, de traditions, voire d'institutions. Tout cela exige en effet que les personnes puissent *partager* leurs intentions et donc *se décentrer d'eux-mêmes* afin de faire émerger des points de référence communs qui sont le début d'un monde objectif, que les symboles permettent de fixer dans le flux de l'expérience subjective. Ce monde objectif naît en particulier dans l'expérience technique. Dewey dit que le langage est « l'outil des outils », en ce qu'il est requis pour qu'un contrôle actif de l'expérience puisse avoir lieu, et notamment que des opérations d'observation et d'inférence puissent être répétées et vérifiées.

> Tout ce qui est utilisé comme outil exhibe distinction et identification. Le feu, existentiellement, brûle ; mais le feu qui est employé à cuisiner et garder au chaud, surtout après que d'autres choses, comme frotter des bâtons l'un à l'autre, aient servi de moyen pour le générer, est une existence ayant une signification et une essence potentielle. La présence de flammes, de terreur ou d'inconfort n'est plus toute l'histoire ; une occurrence est maintenant un objet. […]
>
> Comme le fait d'être un outil, ou encore d'être utilisé comme moyen de conséquences, c'est avoir et procurer signification, le langage, étant l'outil des outils, est la mère chérissante de tout ce qui peut avoir sens (*the cherishing mother of all significance*). Car tout autre instrument et agent, les choses qu'on conçoit usuellement comme appareils, agents, mobilier, ne peuvent naître et se développer que dans des groupes sociaux rendus possibles par le langage.

Ce dernier texte montre le lien étroit que réalise Dewey entre sociabilité humaine, technique et langage. Les premiers systèmes de significations sont techniques et culturels, et leur matrice commune est le langage.

Le rôle éminent du langage provient de ce que les symboles – mots, dessins, monuments – ont des propriétés extraordinaires. Contrairement aux signes naturels, ils ne renvoient à rien naturellement et leur signification doit être instituée. Par ailleurs, ils peuvent en général être invoqués à volonté (proférés, écrits, pensés), ce qui les rend aisément manipulables – raison pour laquelle les monuments sont relativement moins souvent conçus comme des éléments de langage. Mais surtout, l'invocation du symbole dans l'expérience concrète ne crée pas en tant que telle l'attente d'une autre expérience, comme la fumée créée celle du feu. Il ne fait que la *suggérer*, selon l'expression proposée par Dewey, ou encore l'*évoquer* ;

24. *LTI*, p. 27-28.

il a pour fonction de *diriger* l'attention vers certains aspects de la situation ou vers certaines actions possibles[25]. Afin de créer une attente proprement dite, le symbole doit être référé à l'expérience présente, comme lorsqu'on crie « Feu à l'horizon ! » en pointant le doigt dans une direction. Il y a ici une assertion, marquée par la présence d'un déictique. On affirme que quiconque regardera dans cette direction verra une fumée, ou que quiconque se trouve à proximité de cette fumée éprouvera la chaleur du feu, etc. Mais l'expression « feu à l'horizon » en tant que telle est une simple *signification*, pointe seulement vers ces règles opératoires. La signification n'affirme jamais rien de l'expérience présente, mais indique les règles qu'on peut tenter d'y appliquer.

Cette désactivation du rapport immédiat à l'expérience est la source même de la puissance des significations, car elles peuvent de ce fait être invoquées et manipulées à volonté par le biais des symboles. Grâce à elles, l'investigation peut suggérer ou évoquer des inférences entre expériences par le biais de relations entre significations, sans pour autant s'engager aux attentes concrètes sur lesquelles elles se fondent, et donc *peut* par là les mettre à distance afin de les vérifier, d'en identifier les éventuelles conditions d'invalidité, de les corroborer avec d'autres inférences, etc.[26]

Il faut noter qu'il y a ici une double puissance du langage. À un premier niveau, les enchaînements de significations, que sont les raisonnements, permettent de « répéter » mentalement une action avant d'agir, au sens théâtral du terme (*rehearse*)[27]. Lorsque les conséquences représentées ne sont pas celles qu'on espère, on peut ainsi modifier sa conduite par avance. À un second niveau, le langage lui-même, comme système de significations, se rend passible d'examen critique : les inférences qu'il suggère peuvent être critiquées et modifiées : cela est l'objet de cette investigation critique, de second ordre, qu'est la démarche scientifique, sur laquelle nous revenons plus loin.

Les significations peuvent simplement évoquer les expériences sans pour autant générer une attente à leur égard, car fondamentalement elles désignent *les opérations à mener* pour avoir telle expérience, qu'il s'agisse d'une action physique, d'une observation, voire d'une simple attention à tel ou tel aspect de la situation. Elles ont le statut de *règles* : « *Si telle opération est réalisée, alors telle*

25. *LTI*, p. 114.
26. *LTI*, p. 250.
27. *LTI*, p. 63.

expérience s'ensuivra ». Vérifier une signification dans l'expérience, que telle chose *est le cas* (par exemple que la température de la pièce est de 25°), c'est simplement exécuter certaine une opération afin d'éprouver une certaine expérience (consulter le thermomètre et vérifier que la hauteur du mercure se situe à la gradation 25). On se rappelle que la source de ces règles réside, de manière ultime, dans le fait que toute qualité, même la qualité sensible la plus élémentaire, est une certaine réponse de l'environnement à une certaine action générique, que l'organisme a appris a discriminer parmi d'autres réponses possibles.

Ainsi les noms communs désignent des espèces (*kinds*) qui sont des conjonctions d'opérations possibles :

> Les propositions « Ceci a un éclat vitreux ; ne peut pas être rayé par un couteau ; raye le verre ; n'est pas liquéfié par un chalumeau ; se casse en des fractures conchoïdales » sont des propositions qui, prises séparément, énoncent des modes particuliers de changement. Appliqués concurremment et cumulativement, ils donnent l'ensemble des traits conjoints décrivant l'espèce quartz[28].

Les significations ne peuvent exister qu'en relation les unes avec les autres. Les propriétés du quartz dénotent des opérations qui font passer un matériau d'un état à un autre, ces états étant eux-mêmes autant d'expériences potentielles. Par exemple « fractures conchoïdales » signifie à son tour une certaine forme des morceaux qu'il est possible de comparer à celle d'une coquille, ou dont peut mesurer une certaine courbure et un certain poli. C'est donc toujours des signes d'opérations, et des changements qu'elles opèrent dans l'expérience, qui sont mises en relation les unes avec les autres.

Il convient ici de noter que Dewey réserve le terme *inférence* à l'action d'évoquer et d'attendre une certaine expérience à partir d'une autre. Elle se situe donc sur le plan de l'expérience, et non des significations. Sur ce deuxième plan, Dewey parle de *relations*, qui ne sont rien d'autre que les *règles* contrôlant l'inférence. On se rappelle en effet[29] qu'en mathématiques les règles peuvent être (et sont en général) lues comme des relations. Nous utiliserons donc les deux termes pour désigner ces significations, en privilégiant en général celui de *règle*.

28. *LTI*, p. 291.
29. Voir plus haut, § 149.

§ 171. Propositions empiriques et universelles

Les relations servent à contrôler les inférences, à les fonder, et éventuellement à en découvrir de nouvelles. Dewey distingue deux formes à ce rôle.

La première fonction d'une relation, et notamment d'une proposition, est d'orienter l'investigation dans son analyse de l'expérience, en explicitant les inférences qu'on doit y opérer. La proposition *« Ceci est du quartz »* établit une relation entre un objet matériel situé dans l'expérience et l'ensemble des attentes que nous venons d'évoquer, et qui permettent, soit de vérifier la proposition – si on veut s'assurer qu'il s'agit bien de quartz, soit d'indiquer les possibilités ouvertes à l'action ou à la réflexion – ce qu'on peut faire avec du quartz.

Ce type de propositions doit pour cela contenir une référence à l'expérience. Celle-ci peut être explicite, comme lorsqu'on utilise un déictique ou un nom propre, ou un complément circonstanciel. Cependant ces termes, comme *ceci*, *ici*, *Jean*, *Athènes*, ou encore *Au troisième siècle avant notre ère*, par eux-mêmes ne se réfèrent à rien ; ils donnent des indications qui permettent au locuteur de situer la proposition dans son expérience, avant qu'il en puisse considérer la vérité. Comme le dit Dewey, *ceci*, rencontré dans une proposition, désigne d'abord un problème à résoudre, qui est d'identifier à quoi il se réfère. Il est utile ici de remarquer que Dewey critique fortement les exemples artificiels de propositions qu'on trouve dans les manuels de logique, comme *« Socrate est un homme »*, car une proposition n'acquiert de sens complet que dans le cours de l'investigation qui la motive. Ainsi, par exemple, *« Il y a un homme ici »* est une proposition qui a un sens précis dans le contexte d'un endroit plongé dans l'obscurité, et d'un locuteur qui ressent une présence. Il s'agit là d'une *inférence* qui est formulée afin d'inviter à sa vérification. Dans un autre contexte, elle peut être proférée par un promeneur dans la forêt, à qui des gardes demandent s'il a vu passer quelqu'un, et qui pointe un cabanon. Cette proposition est une observation pour le promeneur, mais elle contient une inférence – qu'il s'y trouve toujours. Les gardes doivent également réaliser d'autres inférences – que le promeneur dit vrai et que la personne est celle recherchée. C'est la situation tout entière qui est modifiée par la proposition, ce que Dewey explicite en disant que ce type de propositions permettent d'identifier *ce qui est en question*, le sujet de l'investigation (*subject-matter*).

Il y a également des propositions implicitement référentielles. Dire *« les Athéniens sont fiers »* suppose qu'on puisse localiser cette communauté dans le temps

et l'espace (parle-t-on des Athéniens d'aujourd'hui ou de l'âge classique ?). Mais même une proposition aussi anodine que *« les cygnes sont blancs »* contient une référence à un concept d'histoire naturelle, qui se réfère à des êtres existants dans un certain temps et lieu de l'univers. Ces propositions indiquent des inférences qu'il est possible de faire si, dans l'investigation, on a affaire à des Athéniens ou à des cygnes. Elles peuplent l'arrière-plan de notre expérience de contraintes et de possibilités d'action.

Il y a une seconde forme de relations entre significations, qui n'a pas trait aux inférences qu'on peut faire dans l'expérience, mais aux liaisons qu'il y a entre les seules opérations signifiées. Par exemple, dire que *« qui monte devra descendre pour revenir »* est une relation inhérente aux *significations* de montée et de retour – au moins lorsqu'elles sont entendues dans la topologie de notre espace habituel. Il ne s'agit donc pas d'une inférence à proprement parler, mais d'une simple explicitation des contraintes inhérentes à l'opération conjointe de ces deux significations. Ou encore, dire que *« la distance d'un trajet retour est identique à celle d'un aller »* pointe vers l'équivalence de la mesure d'une distance dans les deux sens ; elle porte sur ce que signifie mesurer, et non sur une connexion de choses dans l'expérience. Il se pourrait d'ailleurs qu'empiriquement cette assertion soit fausse, si par exemple il y a des sens interdits sur le parcours : mais une telle remarque n'a rien à voir avec ce dont il est question ici. Un autre exemple encore, pour sortir du domaine du voyage, est la proposition que si quelques chevaux ont des harnais, alors quelques harnais sont sur des chevaux. Cette affirmation n'a rien à voir avec l'existence de chevaux ou de harnais, mais avec l'équivalence de trouver un cheval harnaché et un harnais sur un cheval. Ces exemples montrent que Dewey ne se situe pas ici dans la perspective kantienne des propositions analytiques – il n'emploie pas d'ailleurs le terme – car il ne s'agit pas d'une simple reformulation propositionnelle des contenus d'une définition des termes en jeu, mais bien d'une analyse opérationnelle, qui tomberait plutôt du côté des propositions synthétiques *a priori*.

Avant d'aller plus loin, il est utile de récapituler les trois rapports que la sémiotique de Dewey distingue : 1) la *connexion* entre objets de l'expérience, *qui seule est l'objet d'inférence* ; 2) la *référence* qu'a une signification à l'expérience, par le biais d'une ou plusieurs opérations de vérification qu'elle indique ; 3) la *relation* qu'entretiennent entre elles les significations[30]. Ces dernières sont de deux types :

30. *LTI*, p. 60-61.

soit elles indiquent qu'une telle relation peut être elle-même vérifiée dans l'expérience, ce qu'on appelle notamment les propositions empiriques, soit qu'il y a un rapport interne entre les opérations par lesquelles on les vérifie, ce qu'on appelle les propositions universelles.

10.2 L'investigation et le jugement

La disponibilité d'un réseau de significations comme le langage permet à la réflexion de reconnaître une situation problématique comme telle, et de l'instituer comme problème[31]. Il s'agit là d'une étape intégrante de sa résolution par l'intermédiaire des significations. Ce moment permet en effet de *déterminer* les termes du problème qui, eux, sont considérés comme fermes, c'est-à-dire non problématiques. Il s'agit des faits qui peuvent être observés et servir de base à sa résolution. Dewey appelle ce moment l'observation, mais il ne faut pas restreindre cette opération à des activités perceptuelles : il peut s'agir également d'hypothèses admises ou de pré-conceptions par lesquelles on comprend la situation initiale. Tout problème évoque une suggestion ou plusieurs suggestions pour sa résolution, car toute signification par laquelle on le décrit dénote d'emblée des possibilités d'action. Une suggestion qui est examinée devient une idée, et cet examen est le raisonnement, qui doit la représenter symboliquement afin de la manipuler. Cette idée peut être considérée immédiatement comme adéquate, et l'investigation est alors terminée. Cependant, une attitude réflexive exige d'inspecter cette idée dans toutes les relations qu'elle contient avec les conceptions de départ, et avec le système de conceptions dans lequel elle apparaît. Ces relations indiquent des manières d'éprouver son adéquation à la situation, elles indiquent des opérations possibles :

> Les faits observés aussi bien que les idées considérées sont opérationnels. Les idées sont opérationnelles en ce qu'elles instiguent et dirigent d'autres opérations d'observation ; elles sont des propositions et des plans pour agir[32].

Cet examen provoque de nouvelles observations et de nouveaux raisonnements, qui permettent de ré-exprimer le problème et de parvenir à une nouvelle compréhension de la situation initiale, où un nouveau raisonnement s'impose – et ce mouvement itératif est reconduit jusqu'à ce que la solution apparaisse comme évi-

31. *LTI*, p. 111.
32. *LTI*, p. 116.

dente dans la ré-expression finale du problème et du raisonnement[33]. Ce moment est le jugement qui clôt l'investigation. La situation apparaît comme « unifiée », selon l'expression de Dewey, c'est-à-dire qu'il n'y a plus désaccord entre le sujet et son environnement.

> L'ordre nouveau des faits suggère une idée modifiée (ou hypothèse) qui occasionne de nouvelles observations, dont le résultat détermine à nouveau un autre ordre de faits, et ainsi de suite jusqu'à ce que l'ordre existant soit à la fois unifié et complet. Dans le cours de ce processus sériel, les idées qui représentent des solutions possibles sont testées ou 'prouvées'[34].

L'investigation comprend ainsi cinq moments qui ne sont pas tant successifs que solidaires :

1. la reconnaissance d'une situation problématique,
2. la position du problème,
3. la suggestion de la solution,
4. son élaboration et sa vérification dans le raisonnement, et
5. le jugement ou la décision, qui clôt l'investigation.

Nous allons nous attacher dans ce qui suit aux deux moments proprement discursifs de l'investigation, d'abord à la position du problème qui mobilise des *moyens matériels et procéduraux* (§ 172), puis au raisonnement qui construit proprement la solution, ou plutôt la *reconstruit* à partir de la suggestion trouvée (§ 173). Nous étudierons ensuite l'investigation du point de vue de son résultat, le jugement, qui est essentiellement une décision qui modifie l'expérience (§ 174).

Cet exposé permet ainsi de répondre à la première partie de notre seconde question concernant la nature commune des jugements pratiques et théoriques. La réponse que lui avait apportée Martin-Löf, étudiée au chapitre précédent, peut être par là éclairée et replacée dans un contexte plus large : elle décrit la preuve comme l'acte idéal du jugement, qui est reconstruit dans la délibération. Cela nous permet, à la section suivante, de répondre à la seconde partie de notre seconde question, concernant l'objet des activités mathématiques.

§ 172. Moyens matériels et procéduraux

Toute investigation s'inscrit dans un système de significations, dont elle mobilise les *moyens matériels et procéduraux*. Il est utile, pour introduire ces termes

33. Voir à ce sujet l'analyse de (Madelrieux 2016, p. 119).
34. *LTI*, p. 117.

importants dans la *Logique* de 1938, de repartir d'une analogie que Dewey établit entre la forme du jugement pratique et la structure sujet / prédicat de la proposition. Dans le cas du jugement pratique, le sujet est le problème de l'investigation, qui est de modifier la situation présente de manière à aboutir à celle désirée. Le prédicat du jugement, *c'est le plan d'action* lui-même, c'est-à-dire la solution à laquelle aboutit l'investigation. Cette solution porte tout autant sur cette construction que sur l'identification initiale du problème dans l'expérience. Il faut, avant d'établir un plan d'action, d'abord reconnaître *ce qui est en jeu* et *interpréter* la situation de sorte qu'elle puisse être décrite dans le système de significations dont on dispose, et qui va fournir les matériaux de cette construction. On peut par exemple reprendre l'exemple très simple de la banane accrochée au plafond, qu'on peut décrocher grâce à un bâton si on monte sur une chaise[35]. L'investigation doit commencer par reconnaître les éléments pertinents dans la situation et ce qu'on peut faire avec : une chaise qu'on peut déplacer et sur laquelle on peut monter, un bâton qui permet d'atteindre des choses éloignées, une banane qui satisfait la faim. Cela est la partie de l'investigation qui *identifie* le sujet du jugement. Sa partie prédicative consiste à monter un plan d'action qui enchaîne ces moyens. Cette partie-là travaille à l'intérieur de l'espace de problème qui a été décrit, c'est-à-dire qu'elle ne fait que relier entre elle les significations données. Dewey appelle moyens matériels les propositions qui permettent d'attacher des significations à des éléments de l'expérience – que la chaise a quatre pieds, une assise et un dossier, par exemple, qu'un bâton est un morceau oblong de bois, etc. Les moyens matériels disent également quelles opérations on peut faire avec ces objets, s'asseoir sur la chaise, ou monter sur elle, saisir le bâton, etc. Quant aux moyens procéduraux, ils désignent les manières d'agencer ces opérations entre elles afin de résoudre le problème, comme par exemple l'enchaînement des actions de prendre le bâton, monter sur la chaise, frapper la banane avec le bâton, etc.

Il y a bien sûr un va-et-vient entre ces deux pôles de la réflexion : si le plan d'action est bloqué, on va rechercher dans la situation si d'autres objets ne pourraient pas être utiles, par exemple une balle qui se trouve là. Et si on remarque réciproquement, par des observations complémentaires, qu'il y a d'autres déterminants à la situation, par exemple que la chaise est cassée, il faut revoir le plan d'action en

35. Cet exemple a été popularisé par Newell et Simon dans leurs études sur le programme GPS (voir ci-dessus, § 64, et annexe A). Ils s'étaient inspirés des expériences faites par le psychologue de la forme Wolfgang Köhler avec des chimpanzés dans les années 1910.

conséquence.

Dans le cas d'un procès, le sujet du jugement (le problème à résoudre) est la situation créée par un crime, qui exige réparation. Le crime est décrit par l'ensemble des faits qui auront été établis. Le prédicat est le raisonnement juridique qui mobilise différents articles de loi et de jurisprudence afin de qualifier ces faits du crime légalement. La sentence finale est l'attribution de ce raisonnement à cette situation, de manière à la résoudre – à ce que réparation soit obtenue. Bien sûr, il ne s'agit pas d'une proposition simple, mais d'une argumentation complexe qui entrelace les faits et les raisons, cependant, la fonction logique de la copule se trouve, selon Dewey, dans cette attribution référentielle d'une signification (complexe) à une situation (complexe).

Si on prend un exemple scientifique, comme le problème du calcul de la vitesse d'une galaxie, le sujet du jugement est constitué par les faits expérimentaux établis ainsi que par les hypothèses théoriques admises. Ces faits sont eux-mêmes le résultat de procédures complexes d'observation, notamment la mesure des raies d'interférence du spectre électro-magnétique, qui requiert des instruments techniques. Des hypothèses peuvent être faites, mais elles doivent être admises par la communauté scientifique, comme celle que la taille de la galaxie étudiée est similaire à celles de même type. On voit ici que la situation n'est pas constituée de faits « bruts », mais qu'elle peut elle-même résulter d'un complexe très vaste de jugements antérieurs, qui est ici le corpus même de la science. Quant au prédicat du jugement, il s'agit du résultat ainsi que de sa preuve complète (observations, principes, hypothèses, calculs) organisée. Les moyens matériels sont ici les instruments d'observation, les théories mathématiques utilisées, les principes physiques ; les moyens procéduraux sont les méthodes par lesquelles tous ces moyens sont mobilisés (par exemple avec les approximations mentionnées) pour agencer la preuve.

La dualité des moyens matériels et procéduraux n'est pas limitée à leur forme discursive, mais elle imprègne toutes les pratiques élémentaires de la vie[36]. Dans les activités productives, les moyens matériels représentent ainsi les outils et les matériaux, et les moyens procéduraux les techniques qui les utilisent[37]. Ce cas particulier fait apparaître deux choses importantes. D'abord ces deux classes de moyens sont toujours adaptés l'un à l'autre : on a les techniques que permettent nos outils, et nos outils sont conçus en fonction de nos techniques. Cette adap-

36. *LTI*, p. 81.
37. *LTI*, p. 383.

tation est dynamique, en ce que l'investigation a la possibilité de les modifier en fonction de ses besoins. Pour Dewey, l'investigation n'est jamais enfermée dans un espace de problème rigide pour trouver ses solutions, mais est toujours capable de reconfigurer ses espaces de problème au cours de sa recherche[38]. Ensuite, l'essence des moyens procéduraux est d'instaurer un *ordre sériel*, qui caractérise aussi bien les procédures techniques que le discours rationnel en général[39]. Cet ordre est imposé par le fait même que toute enquête est *dirigée* vers la résolution du problème en jeu, ce que Dewey appelle même le « principe de direction[40] ». Il le compare à celui qui ordonne les degrés d'une échelle, et l'oppose à l'ordre matériel qui organise les faits et les données, et que Dewey décrit comme l'ordre d'un ensemble plutôt que d'une série – nous dirions aujourd'hui une *structure*.

Ces exemples permettent de comprendre qu'un des rôles d'un système de significations – le langage courant pour les cas pratiques élémentaires, le droit, les théories scientifiques – est de fournir à l'investigation les concepts adéquats pour les représenter et construire ses procédures de résolution. Il lui évite d'avoir à construire ses propres concepts, en lui permettant de les puiser dans un réseau de relations déjà établies et contrôlées.

La dualité des *moyens procéduraux* et *moyens matériels* (*procedural and material means*), traverse ainsi la *Logique* de 1938 tout entière et s'exprime également par d'autres couples de termes : existentiel et idéationnel (*ideational*), factuel et conceptuel, description et définition, etc.[41] Cette dualité joue ainsi un rôle fondamental dans l'armature conceptuelle de l'ouvrage, et toute la logique classique est interprétée à son aune. Au niveau de la structure de la proposition, comme nous venons de le voir, elle explique la dualité du sujet et du prédicat. Au niveau des typologies de propositions, elle explique la dualité entre propositions empiriques et universelles, que nous avons vue à la section précédente. Enfin, le syllogisme classique lui-même dénote selon Dewey cette structure fondamentale de la réflexion, où la majeure représente une solution disponible (c'est-à-dire déjà construite) dans le système pour un problème donné (par exemple : *« Lorsqu'on est malade, il faut voir un médecin »*), la mineure, l'identification de la situation à ce problème (par exemple : *« Je suis malade »*), et la conclusion la décision d'appliquer la solution

38. *LTI*, p. 392.
39. *LTI*, p. 117, 384.
40. *LTI*, p. 312-313.
41. Voir notamment (*LTI*, p. 272, 275, 283).

proposée à la situation (« *Je dois voir un médecin* »)[42]. Bien entendu, cette manière de présenter la résolution de problème est caricaturale, et Dewey la dénonce comme telle. Les choses se passent rarement comme cela, sauf dans les cas où l'on se trouve dans un problème paradigmatique prévu par la théorie, comme par exemple, la construction d'un triangle équilatéral dans les *Éléments*. Mais, dans le cas général, l'identification de la situation se fait de manière désordonnée, plusieurs significations du système y étant progressivement reconnues une à une. La construction de la majeure est également complexe, car il faut relier entre elles ces propriétés du problème en reconnaissant un enchaînement possible d'opérations les concernant, et l'ensemble de ce double travail est itératif.

Les moyens procéduraux disent comment agencer entre eux les moyens matériels pour résoudre un problème, par exemple chauffer le métal *afin de* le faire passer dans un moule qui lui donnera la forme requise une fois refroidi. Ils représentent les manières de habituelles de faire, les méthodes, les heuristiques. Par opposition, les moyens matériels énoncent donc les règles élémentaires qui régissent les objets du problème, et dont se servent les moyens procéduraux. Ils disent quelles inférences on peut légitimement réaliser à leur égard, par exemple que le métal devient liquide quand on le chauffe, et qu'un liquide prend la forme de son contenant.

Il ne faut cependant pas voir les moyens matériels comme des matériaux « inertes » sur lesquels agiraient les moyens procéduraux. Comme ne cesse le répéter Dewey, les faits sont opérationnels en ce qu'ils *suggèrent* des opérations possibles. Il y a dans l'idée du métal qu'il peut être chauffé, et le marteau, selon l'adage consacré, invite à regarder tout problème comme un clou. La procédure que nous avons citée plus haut *ne se déduit pas* des deux propositions matérielles citées après, mais se construit à partir d'elles. Si les moyens matériels étaient inertes, nulle procédure ne pourrait jamais être construite les concernant. Ils signifient déjà au contraire quelles actions élémentaires peuvent et doivent être prises en compte pour déterminer un plan. Comme le dit très bien cette formule lapidaire de Dewey : « La poudre est ce qui explosera sous certaines conditions[43] ».

Ces moyens ne sont pas non plus « matériels » en ce qu'ils désigneraient un terme ultime et inanalysable de la pensée, comme les empiristes anglais par

42. *LTI*, p. 167.
43. *LTI*, p. 132.

exemple ont pu parler des données des sens comme « matériau » de la pensée. La critique de cette position par Dewey a déjà été mentionnée à la section précédente. Dewey les appelle tels parce qu'ils permettent de *reconnaître* la « matière » ou sujet du problème (Dewey dit : *subject-matter*), c'est-à-dire de le cadrer à l'aide d'un système de significations.

§ 173. La reconstruction de l'investigation dans le raisonnement

La position d'un problème, selon Dewey, suggère ainsi des solutions, ou plutôt des pistes de solutions, car telle est la fonction des moyens matériels, fournir des outils pour agir. Dès que j'ai posé un système d'équations différentielles, je sais qu'il y a certaines transformations que peux y opérer, en reconnaissant par exemple des formes similaires à des systèmes déjà résolus. Des moyens procéduraux sont par là également suggérées, c'est-à-dire des méthodes que je peux mettre en œuvre. La solution est alors trouvée, ou j'aboutis à un autre système d'équations, peut-être simplifié. Il s'agit là d'un nouveau problème, qui peut me suggérer à son tour d'autres opérations à réaliser, et ainsi de suite. Pour prendre un autre exemple, si je dois planifier l'approvisionnement de mes entrepôts de distribution pour le mois à venir, la suggestion immédiate est de reconduire le plan du mois précédent. En vérifiant cependant la capacité de mes usines, je m'aperçois que deux d'entre elles seront en maintenance. Je dois alors chercher des compléments de production ailleurs, ce qui me suggère d'aller vérifier les disponibilités d'usines plus éloignées dans mon réseau logistique. Dans ces démarches itératives, le raisonnement est chaque fois la réalisation des opérations symboliques suggérées par la piste considérée : dans l'exemple mathématique, il s'agit de réaliser les transformations algébriques sur mon système d'équations pour voir ce que cela donne ; dans l'exemple logistique, de vérifier dans les rapports de prévisions les disponibilités de mes usines. Toutes ces opérations sont symboliques, mais pas nécessairement mentales.

Ainsi, dans l'investigation, l'étape du raisonnement a pour rôle d'activer les moyens matériels et procéduraux du problème afin d'examiner les solutions qu'ils suggèrent. Son rôle n'est donc pas tant de trouver la solution, que de la vérifier formellement (sans recours à l'expérience). On retrouve là l'idée kantienne que le raisonnement est un jugement complexe *qui régresse* vers ses prémisses, plutôt qu'une composition de jugements entre eux pour en déduire un nouveau[44]. Le

44. Voir plus haut § 134.

raisonnement a donc un double rôle, clairement exprimé dans ce passage :

> Le raisonnement a le même effet sur une solution suggérée que l'observation plus serrée et approfondie du problème d'origine. [Parfois,] l'acceptation de la suggestion dans sa forme initiale est bloquée par un examen plus approfondi. Les conjectures qui semblent plausibles à première vue se révèlent souvent inadéquates ou même absurdes quand leurs conséquences complètes sont retracées. Et même lorsque l'explication rationnelle des implications d'une supposition ne mène pas à son rejet, elle en développe l'idée dans une forme qui est plus pertinente au problème. […] Le développement d'une idée par le raisonnement permet *a minima* de fournir les termes intermédiaires ou incidents qui lient en un tout cohérent des extrêmes apparemment en désaccord[45].

On voit ici que le raisonnement a d'abord un rôle de vérification de la suggestion de solution, qu'il faut distinguer de la validation expérimentale du jugement, celle-ci impliquant l'observation empirique. Cette vérification développe les conséquences d'une idée, qui peuvent la bloquer en regard des données du problème, ou au contraire en indiquer les relations précises qui permettent de le transformer en une solution. Le raisonnement *construit* donc la solution. En fonctionnant de manière itérative avec les suggestions de solution (étape 3 de l'investigation) et la reformulation du problème, éventuellement grâce à de nouvelles observations empiriques (étape 2), il peut être amené à reconstruire le problème tout entier jusqu'à ce qu'y apparaisse une solution satisfaisante. Cette solution (et c'est là le second rôle du raisonnement) n'exhibe pas seulement le point d'arrivée qu'il est possible ou convenable d'atteindre, mais également la séquence des moyens pour ce faire, soit dit en termes imagés, le chemin pour y parvenir – ce que nous appelons, par anticipation, *procédure*[46]. Ce terme doit être pris ici en son sens le plus large, qui peut aussi bien être une procédure de preuve mathématique, qu'une procédure d'actions à mener pour résoudre un problème de la vie courante.

Il ne faut donc pas confondre la procédure finale avec l'investigation elle-même. Nous avions relevé cette ambiguïté chez Mäenpää, qui tentait de montrer l'identité des deux notions à travers celle intermédiaire d'analyse[47]. L'investigation, dit souvent Dewey, *reconstruit* le problème tout entier. Le problème tel qu'il se pose à la fin ne ressemble plus à la manière dont il se présentait au début. Alors il n'était que partiellement compris, il y avait du vague, des éléments accidentels apparais-

45. (Dewey [1910] 1997, p. 76).
46. Voir plus loin, § 175.
47. Voir plus haut, § 164.

saient comme saillants, alors que des éléments décisifs étaient cachés, des pistes de résolution suggérées qui étaient des cul-de-sac, etc. L'investigation produit, à son terme, une nouvelle description du problème, une *spécification* finale à partir de laquelle une procédure peut être nettement et directement détaillée. Mais cette procédure ne dit rien des itérations successives, de l'exploration, des doutes, des tentatives avortées, qui ont caractérisé l'investigation[48]. Elle décrit simplement les relations qui relient le problème, tel qu'il apparaît dans sa forme finale, avec sa solution.

L'argumentaire que développe le juge, et qui motive sa décision, ne se confond pas avec le procès, ni même avec ses minutes, puisque celui-ci a été constitué de débats, d'interrogatoires, de re-dites et d'arguments, dont le juge n'a repris que ce qui était utile à l'élaboration de son jugement. De la même manière, le plan d'action ne se confond pas avec la délibération qui le produit, et ni la méthode scientifique de calcul ou d'expérimentation avec son élaboration.

La procédure est donc la *reconstruction idéale*, dans le domaine des significations, de l'investigation elle-même. Elle indique la démarche à suivre qui permet de parvenir au même résultat que l'investigation en évitant la difficulté et le hasard de la recherche. Ce simple fait explique pourquoi les études empiriques ne décèlent que rarement des déductions, ni même des syllogismes dans les processus cognitifs. Un syllogisme comme : *« Lorsqu'on est malade, il faut voir le médecin. Or je suis malade. Donc, etc. »* ne traduit pas du tout le processus de réflexion de quiconque se demande s'il doit aller voir un médecin ou non, car cela voudrait dire qu'il n'y a pas eu d'investigation à proprement parler, mais une sorte d'habitude automatique, comme une réponse à un stimulus[49]. Le syllogisme pourrait être plutôt formulé dans le conseil d'une personne à laquelle on demanderait un avis, ou dans l'explication qu'on lui donnerait de notre résolution. Cette ligne d'interprétation rejoint la thèse exprimée récemment par Sperber et Mercier (2017), que la raison a une fonction essentiellement pédagogique et justificatrice – sans pour autant qu'il faille conclure de cela, avec ces auteurs, que la rationalité n'a pas cours dans le processus de l'investigation, et que celle-ci serait entièrement commandée par des habitudes cognitives, et donc biaisée. La procédure rationnelle, ou encore la démonstration qui enchaîne rigoureusement les significations, est bien plutôt le terme que le moyen, l'enjeu d'un combat contre nos habitudes inférentielles que

48. *LTI*, p. 124, 161.
49. *LTI*, p. 167.

l'investigation vise à contrôler.

Ce mouvement de reconstruction n'est pas nécessaire à toutes les investigations. L'investigateur se contente, la plupart du temps, de la solution trouvée qu'il applique immédiatement à la situation problématique. Cette reconstruction est par contre essentielle à certaines formes d'investigation, par exemple celles du droit ou des sciences, qui s'engagent à produire des jugements en bonne et due forme, c'est-à-dire qui contiennent leur propre justification[50]. La reconstruction de l'investigation est donc comme une seconde investigation, en posture réflexive vis-à-vis de la première, qui vise à déterminer quels furent les éléments déterminants pour la formation du jugement final, quels furent les jugements intermédiaires, significatifs, etc.

Dans cette seconde investigation on porte également en général attention à la *généralisation* possible du problème, si celui-ci est paramétrique. Dans ce cas le mouvement de reconstruction se déroule à un niveau encore supérieur de réflexion, puisqu'il ne s'agit pas seulement d'observer quelles furent les étapes décisives du jugement, mais de trouver comment et pourquoi ces étapes furent elles-mêmes identifiées, car elles pourraient être différentes dans d'autres cas de figure. En d'autres termes, on s'intéresse ici à l'*heuristique* elle-même, c'est-à-dire au mouvement par lequel le problème a été posé, reconfiguré, etc. Cette reconstruction de second ordre nous intéresse particulièrement, car c'est cette forme d'investigation qui est caractéristique de la programmation.

On retrouve ici les trois classes d'impératifs qu'avaient distinguées Simon dans la résolution de tout problème : les impératifs (A), qui indiquent l'objectif à atteindre, les impératifs (B), qui demandent d'expliciter le plan d'action pour ce faire, et enfin les impératifs (C), qui demandent comment constituer, d'une manière générique, un tel plan d'action[51]. Ces trois classes sont précisément articulées et expliquées dans la théorie de l'investigation de Dewey : (A) désigne un impératif qui dirige l'investigation simple, (B) celui qui dirige la mise en procédure de l'investigation, et (C) celui qui vise à décrire la méthode d'une telle mise en procédure, pour une classe plus ou moins générale de problèmes similaires à celui initialement résolu.

Nous définissons enfin (et cette définition est ici la nôtre) la *ratiocination* comme une investigation qui se déroule par l'exécution scrupuleuse des règles

50. Nous précisons ce point au développement suivant.
51. Voir plus haut, § 65.

d'une procédure déjà connue. Nous avons choisi à dessein ce terme vieilli, occasionnellement employé par Dewey, en laissant de côté le sens péjoratif qu'il a aujourd'hui, parce qu'il évoque bien la forme de réflexion que nous visons ici, celle qu'on accomplit pas à pas et dans le scrupule des règles d'inférence prescrites. Il s'agit là d'une d'investigation plus large que celle dénotée par le terme *calcul*[52], pour au moins deux raisons – en mettant d'emblée de côté son association initiale aux catégories du nombre et de la quantité. D'une part, toutes les procédures n'ont pas l'exactitude et l'infaillibilité des règles de calcul. D'autre part, le calcul procède par transformation de données d'entrée en données de sortie sans qu'aucune interruption soit possible, ni pertinente, avant l'atteinte de son résultat. Or de la même manière que l'investigation se fait en constante interaction avec la situation, une ratiocination peut en général être également ouverte à des interactions réglées. Ainsi, lorsqu'un élève reprend les instructions de son professeur pour effectuer un exercice, qu'un contremaître organise le travail d'un atelier conformément aux consignes de l'ingénieur, ou qu'un agent commercial, dans son dialogue avec un client, applique soigneusement le *script* fourni par sa direction, on peut dire qu'il y a là des cas de ratiocination, et non de calcul.

La ratiocination est une étape dans l'apprentissage, c'est-à-dire dans la résorption de situations problématiques en expériences habituelles, qui ne posent plus problème. Ainsi, l'apprentissage des méthodes de calcul passe par leur exécution « mécanique », et la discipline laborieuse que cela impose, jusqu'à ce qu'elles soient intégrées à nos compétences cognitives et que leur application devienne « naturelle », sans effort, et presqu'inconsciente. La procédure est une étape dans *l'internalisation* de la réponse adéquate à apporter à une situation jusqu'ici inédite. Elle remplace l'investigation complexe et coûteuse et rend possible l'émergence d'une pratique habituelle qui, répétée un certain nombre de fois, peut aller jusqu'à devenir un automatisme inconscient.

§ 174. Le jugement comme acte dans l'expérience

Dewey définit le jugement comme l'acte qui clôt l'investigation.

Dans le chapitre de la *Logique* consacré à cette notion et qui s'appelle, de manière significative, *La construction du jugement*, Dewey prend l'exemple d'un procès : le jugement final *a* des conséquences : libérer un prévenu ou l'envoyer en

52. Nous revenons sur le *calcul* au chapitre suivant, section 11.1.

prison, exiger des dommages, etc. On peut également employer les termes *décision, règlement* (au sens de régler une affaire, *to settle*[53]), ou *résolution* pour véhiculer la force énactive que Dewey entend donner au jugement. Par ailleurs, le jugement est *justifié* par l'investigation, il peut se prévaloir du cheminement accompli pour expliquer sa décision – quand bien même celui-ci aurait été sommaire et « déraisonnable » selon nos standards, comme par exemple l'invocation de tabous dans les sociétés traditionnelles pour justifier une manière d'agir[54].

Il y a donc deux aspects à la thèse que le jugement est un acte. D'abord, il s'agit bien d'une action qui intervient dans l'expérience et qui transforme une situation indéterminée en une situation déterminée. Par ailleurs, sa justification est inséparable de la procédure qui récapitule l'investigation. Nous allons développer ces deux points successivement.

Le jugement pratique, c'est-à-dire la résolution à agir conformément à un plan, est exemplaire concernant le premier point. Les significations ici représentent des actions possibles et la modification que chacune entraîne dans l'expérience. L'investigation consiste à les enchaîner en un plan dont elle estime qu'il conduit à la situation désirée, et le jugement clôt cette investigation par la résolution d'agir.

Aussi le jugement doit-il être radicalement distingué de la proposition. Un jugement *peut* être exprimé par une proposition mais il ne s'y réduit pas. Une proposition est une relation entre significations, et donc n'a pas de référence intrinsèque a l'expérience, comme on l'a vu plus haut. Un jugement, au contraire, est motivé par un problème qui survient dans l'expérience et dont il est la résolution. Il ne s'agit pas de juger quelqu'un *in abstracto*, comme on peut le faire dans des études de cas fictives, mais de juger telle personne qui a causé tel tort. Il ne s'agit pas de savoir ce que l'on ferait si jamais telle ou telle situation arrivait, mais de décider quel plan d'action développer dans la situation présente. Le jugement a donc une importance expérientielle directe que la proposition n'a pas :

> Le jugement peut être identifié comme une résolution finale de l'enquête […] En ce sens il se distingue des *propositions*. Le contenu de ces dernières est intermédiaire, il sert à la représentation et il est véhiculé par des symboles ; tandis que le jugement, tel qu'il est finalement rendu, a une portée existentielle directe. […] J'utiliserai le terme d'*assertion* pour désigner le statut logique [de ce contenu qui été préparé pour être final][55].

53. *LTI*, p. 124.
54. *LTI*, p. 68.
55. Texte original : *Judgment may be identified as the settled outcome of inquiry […] In this*

Dans le cours d'une investigation, on peut certes *considérer* des propositions qui semblent pertinentes pour le problème présent. Par exemple, la proposition que le prévenu était à tel endroit au moment du crime peut être avancée ; mais afin de pouvoir devenir une pièce utilisable dans le procès elle doit être établie, c'est-à-dire devenir l'objet d'un jugement partiel qui viendra éventuellement étayer l'argumentaire du jugement final.

La thèse de Dewey est que tous les jugements authentiques sont de tels *actes* qui viennent transformer l'expérience, y compris les jugements scientifiques. Reprenons l'exemple du calcul de la vitesse d'une galaxie. On peut faire ce calcul en guise d'exercice, ou encore pour étalonner un nouvel équipement. Le jugement ne porte pas alors sur le résultat, mais sur la satisfaction donnée par l'élève ou par la machine. Peut-être ce calcul a-t-il été fait par simple *curiosité*. Mais il faut alors qualifier celle-ci afin de pouvoir qualifier le jugement qui la résout : s'agissait-il d'une anomalie que l'astronome croyait avoir détecté ? dans ce cas, le calcul s'inscrit dans une investigation plus large, qui touche à la mise en cohérence ou à la falsification d'un ensemble d'hypothèses scientifiques. S'agissait-il d'utiliser ce résultat afin de calculer d'autres grandeurs physiques, comme la vitesse d'expansion de telle région de l'univers ? Dans ce cas comme dans le précédent, le jugement sert à construire un jugement plus complexe, d'un intérêt certes entièrement « théorique »– nous revenons sur cette question plus loin – mais qui modifie le savoir de l'astronome, en confirmant ou infirmant ses théories, et qui a vocation à être publié, donc à changer plus généralement l'état de la science.

Le jugement est également une action, non seulement en ce qu'il modifie l'expérience, mais en ce qu'il est lui-même un processus, distinct cependant de l'investigation. Dewey rend cela visible par cette analogie surprenante qu'il propose entre le jugement et une organisation industrielle :

> On peut voir [la copule] […] comme ce qui met les contenus sujet - prédicat au travail, à *exécuter* leurs fonctions les uns par rapport aux autres. Dans les entreprises complexes, un plan de répartition des fonctions est généralement établi sur papier. Mais ce plan n'est pas la division réelle du travail. Celle-ci consiste en la distribution effective des facteurs actifs de ce qui se fait dans la coopération des uns avec les autres. […]

sense [it] is distinguished from propositions. The content of the latter is intermediate and representative and is carried by symbols ; while judgment, as finally made, has direct existential import. […] I shall use assertion to designate the logical status [of subject-matter which has been prepared to be final]. (*LTI*, p. 123)

> Le plan peut être exposé et expliqué dans des propositions ; son exposition propositionnelle peut être un outil pour critiquer et réaménager le plan de distribution. Mais la division proprement dite ne peut qu'être énactée. […] Il n'est pas plus une division fonctionnelle du travail qu'un plan d'architecte n'est une maison en cours de construction ou qu'une carte n'est un voyage[56].

Ce qu'essaie de penser Dewey, c'est que les étapes intermédiaires du raisonnement qui conduisent au résultat final sont *conservés* dans le jugement, non seulement à titre documentaire, mais bien comme entités actives. L'origine de cette analogie se trouve quelques paragraphes plus haut du texte cité, où Dewey combat la thèse de Bradley qu'il y aurait une contradiction inhérente à l'idée de jugement, puisqu'il consiste à distinguer dans l'expérience des traits pour ensuite les réunifier (comme on distingue la rougeur de la pomme pour ensuite attribuer la première à la seconde) :

> Puisque [sujet et prédicat] sont fonctionnels et opératoires, il n'y a pas plus de conflit qu'il n'y en a dans le fait qu'au cours de toute activité productive complexe, industrielle ou sociale, s'instituent des divisions du travail qui pourtant sont fonctionnellement liées les unes aux autres. Car elles sont instituées comme moyens de coopération en vue d'un résultat commun unifié. […] Le jugement, comme l'enquête, est temporel. Il est temporel, non pas au sens externe que l'acte de juger prend du temps, mais au sens où son objet subit une reconstitution en atteignant l'état final de résolution et d'unification déterminées qui est l'objectif qui gouverne le jugement[57].

Si le jugement ne se confond jamais avec le résultat brut de l'investigation, comme la résolution de faire telle chose, ou la sentence du condamné, ou un nombre représentant une vitesse, c'est qu'il comprend également sa justification, c'est qu'il est la construction tout entière qui permet d'assurer ce résultat[58]. Comme le dit Dewey :

> À travers une série de significations intermédiaires, une signification est finalement atteinte qui est plus clairement pertinente pour le problème en question que l'idée initialement suggérée. Elle indique les opérations qui peuvent être effectuées pour tester son applicabilité […] En d'autres termes, l'idée ou la signification, lorsqu'elle est développée dans le discours, dirige les activités qui, lorsqu'elles sont exécutées, fournissent son évidence matérielle[59].

Ainsi, la résolution d'agir acquise par le jugement pratique diffère d'un simple coup

56. *LTI*, p. 138.
57. *LTI*, p. 136-137.
58. Voir à ce sujet (Madelrieux 2016, p. 103).
59. *LTI*, p. 115.

de tête *parce qu'*elle se place sous l'égide d'un plan d'action réfléchi, le jugement du tribunal diffère d'une sanction arbitraire par l'argumentation que construit le juge à l'issue du procès, et le résultat scientifique d'une simple estimation par la méthode réplicable qu'il exhibe.

> Un travail scientifique en physique ou en astronomie produit un rapport (*record*) des calculs et des déductions dérivés d'observations et d'expériences passées. Mais c'est plus qu'un rapport ; c'est aussi une indication, une assignation à effectuer d'autres observations et expériences. Aucun rapport scientifique n'obtiendrait considération s'il ne décrivait pas l'appareil au moyen duquel les expériences ont été menées et les résultats obtenus ; non pas que l'appareil soit vénéré, mais parce que cette *procédure* indique aux autres investigateurs comment se mettre au travail pour obtenir des résultats qui concorderont ou non, dans leur propre expérience, avec ceux précédemment atteints, et ainsi confirmer, modifier et rectifier ces derniers. Le résultat scientifique consigné dans un rapport est dans l'effet la désignation d'une méthode à suivre et une prédiction de ce qui sera trouvé lorsque des observations spécifiées seront exécutées[60]. *(Nous soulignons)*

Le plan, qui n'est rien d'autre que le raisonnement, ou le discours ordonné final, est donc cet enchaînement de significations *auquel aboutit* l'investigation et que le jugement attribue à la situation. Hors du jugement, tout raisonnement, constitué de propositions, a le même statut qu'elles : il est entièrement hypothétique, et l'ensemble de ses prémisses matérielles (qui se réfèrent à l'expérience) doivent être chacune l'objet d'une assertion, afin que lui-même puisse l'être.

10.3 Les mathématiques et les moyens procéduraux

Cette section est consacrée à la conception des mathématiques proposée par Dewey. Nous revenons d'abord (§ 175) sur la notion de procédure, très présente dans la *Logique*, avant d'interpréter l'assertion de Dewey que les mathématiques sont des systèmes procéduraux (§ 176). Nous montrons à ce titre que Dewey conçoit déjà les preuves comme des formes de procédures (§ 177). Cela nous amène à nous interroger sur les rapports qu'entretient Dewey avec l'intuitionnisme, d'abord sur le plan historique puis sur le plan logique (§ 178) : il nous apparaît que la théorie des types de Martin-Löf est un des formalismes universels possibles dans lesquelles toute investigation peut se récapituler de manière procé-

60. LW 1:39.

durale.

§ 175. La notion de procédure

Il n'a peut-être pas échappé à la lectrice que Dewey donne un rôle central au terme *procédure* dans l'armature conceptuelle de sa théorie de l'investigation. Sans compter son occurrence l'expression *moyens procéduraux*, il apparaît de manière fréquente dans l'ouvrage (85 fois, à comparer aux 137 et 260 occurrences de *processus* et *méthode*, respectivement) à une époque où il est en train d'acquérir son sens moderne. Peut-on voir là un signe que la *Logique* de Dewey préfigurerait les développements de l'informatique et de la programmation ? Les choses ne sont pas aussi directes que cela.

Le dictionnaire américain Merriam-Webster de 1936 présente le mot ainsi :

> PROCÉDURE, n. 1. Manière ou méthode (*manner or method*) de procéder dans un processus ou une ligne d'action (*course of action*) ; également, une façon (*way*) particulière de procéder. 2. La poursuite d'un processus ou d'une opération ; progrès. 3. Méthode coutumière (*customary method*) de conduite des affaires dans un organe délibératif ; ordre parlementaire ; ainsi, les règles de procédure.
>
> PROCÉDER, v.i. 1. Se déplacer, passer devant ou aller de l'avant, avancer. 2. Provenir ou apparaître comme d'une source ou d'une origine ; venir (de). 3. Continuer d'une manière ordonnée ou réglée ; poursuivre une conception. 4. Juridique. Entamer et poursuivre une action judiciaire. Syn. Voir AVANCER. Antonymes. Régresser, reculer, se retirer ; arrêter[61].

Afin de bien percevoir l'écart entre ces sens dont dispose Dewey et celui qui a cours aujourd'hui, il est utile de se référer à la définition donnée par le même dictionnaire aujourd'hui :

> PROCÉDURE. 1. Une manière particulière d'accomplir quelque chose ou d'agir ; une étape dans une procédure. 2. Une série d'étapes suivies dans un ordre défini et régulier : une procédure légale, une procédure chirurgicale ; un ensemble d'instructions informatiques qui reçoit un nom par lequel leur exécution peut être invoqué. 3. une manière traditionnelle ou établie de faire les choses ; (protocole) un code prescrivant le respect strict de l'étiquette et de la préséance (comme dans les échanges diplomatiques et les affaires militaires)[62].

Ainsi, en 1936, le sens de *procédure* est encore largement dépendant de celui de *procéder*, et donc de la notion générale de progression ; il peut être aisément sub-

61. *Webster's collegiate dictionary* (1936), p. 790.
62. Dictionnaire *Merriam-Webster* en ligne, merriam-webster.com.

stitué à *processus*. Cependant l'accent est déjà mis sur la manière, la coutume et la méthode de procéder, sans doute sous l'influence de la connotation juridique du terme. Son sens actuel, comme description ou prescription rigoureuse d'un processus en étapes unitaires indépendantes, centrale aujourd'hui, n'apparaît pas encore. Cette transformation se voit dans l'évolution croisée de l'usage des termes *procédure* et *procéder*. La forme verbale, en effet, dont la signification a peu évolué, voit sa fréquence relative divisée par 2,5 environ entre 1900 et 2000 selon l'outil Google Ngrams[63]. L'usage de *procédure*, au contraire, dont le sens devient progressivement autonome de celle-ci, explose entre 1900 et 1945, passant de 30 à 115 occurrences par million de mots, pour se stabiliser à ce niveau jusqu'en 1990, où il amorce un déclin rapide.

Son emploi par Dewey en 1938 reflète ces transformations. Il entend en effet *procédure* dans son sens usuel de l'époque, comme progression dans le cours d'une action, et il l'intervertit d'ailleurs parfois avec *processus*[64]. Il utilise le terme dans son sens descriptif général, c'est-à-dire *manière de procéder* et il est alors étroitement associé à la résolution de problème, ainsi lorsqu'il évoque les médecins et les avocats dont la bonne procédure « consiste essentiellement à formuler les bonnes questions[65] » : ici le terme a le rôle très général d'un précepte, ou d'une bonne pratique dans la conduite d'une investigation. Cependant, il l'associe particulièrement au terme *méthode*, avec lequel il apparaît souvent de manière conjointe, et cela dans le contexte de l'investigation scientifique[66]. Dans ce contexte, Dewey parle bien de procédure au sens de l'établissement d'un ordre séquentiel dans la progression du raisonnement ou de l'action, et évoque l'image de « barreaux d'une échelle », qui fait bien référence aux étapes bien distinctes des procédures telles qu'elles sont comprises actuellement.

§ 176. Les mathématiques comme systèmes procéduraux

Les mathématiques ont un statut procédural pour Dewey, ce qui les rapproche des techniques[67], ainsi qu'on le voit par l'affirmation suivante :

> La nature conceptuelle des données matérielles des mathématiques signifie qu'elles

63. Voir notre note concernant l'usage prudent, et simplement tendanciel, que nous faisons de cet outil § 25.
64. Par exemple *LTI*, p. 15,436.
65. *LTI*, p. 172.
66. *LTI*, p. 26, 65, 390, 404, etc.
67. *LTI*, p. 384.

> sont déterminées exclusivement et entièrement en référence à la possibilité d'opérations de transformation, *ces dernières constituant les moyens procéduraux*[68]. *(Nous soulignons)*

Les mathématiques sont des systèmes entièrement procéduraux en ce que leurs moyens matériels ne disposent d'aucunes marques par lesquelles ils peuvent être référés, directement ou indirectement, à l'expérience concrète – ils dénotent de pures opérations possibles, comme compter. Contrairement à une théorie des sciences naturelles, elles ne disposent pas de concepts comme, par exemple, ceux de d'*espèce animale*, de *cellule*, de *masse*, de *charge*, ou d'*énergie*, qui peuvent tous être reconnus ou mesurés dans l'expérience, fût-ce grâce à des procédures d'observation ou de mesures complexes et indirectes. Au contraire les moyens matériels des mathématiques signifient seulement des opérations possibles sur des éléments qui sont définis exclusivement en regard de celles-ci.

> La condition à laquelle doivent satisfaire les contenus mathématiques, qu'il y ait possibilité de transformation (*tranformability*), exige qu'il y ait des « données » déterminées exclusivement et exhaustivement par référence aux opérations et règles d'opérations exécutées ou à exécuter avec et sur elles[69].

La *transformation* est un terme que Dewey introduit spécifiquement dans son développement sur les mathématiques pour caractériser le progrès du discours ordonné (le raisonnement) qui substitue les significations les unes aux autres par des relations contrôlées. Les mathématiques portent donc sur des inférences allant d'une opération possible à une autre : *Si je puis faire ceci et obtenir tel résultat, alors je puis faire cela et obtenir tel autre*, comme par exemple : *Si je puis additionner 2 et 3 et avoir 5, alors je puis additionner 2 et 4 et avoir 6*. Les nombres 2, 3, 5, etc. ici sont définis exclusivement en regard des opérations élementaires instituées sur eux, celles de l'axiomatique de Peano par exemple.

Cette caractéristique explique les propriétés par lesquelles on décrit habituellement les mathématiques. D'une part, elles ne peuvent être infirmées par l'expérience, car elles sont libérées de toute *référence* à celle-ci. On ne reconnaît pas qu'il y a *cinq* chevaux dans un pré comme on reconnaît qu'ils sont noirs, mais on les *compte*. Pour la même raison, il n'est pas possible que ces cinq chevaux et trois de plus qui arrivent ne soient pas huit, parce que cette relation est dérivée directement d'une chaîne de relations portant sur l'acte de compter, et non sur les chevaux.

68. *LTI*, p. 403.
69. *LTI*, p. 402.

Cela explique également l'applicabilité universelle des théories mathématiques, par exemple la théorie des graphes, au sens où leurs résultats peuvent être affirmés dans l'expérience du moment que ses opérations élémentaires peuvent y être décrites. Elle ne dit rien d'autre que : *si telles opérations y sont effectives, alors telles autres le sont aussi.*

D'autre part, pour cette raison même, les mathématiques, comme sciences, c'est-à-dire comme démarche de systématisation des relations entre des opérations pures, se présentent comme des systèmes symboliques « formels », où on peut croire que les symboles n'ont aucune signification, et n'ont de sens que par les règles de syntaxe qui leurs sont imposés. Mais on doit plutôt dire, dans le vocabulaire de Dewey, que les mathématiques n'ont aucune *référence* c'est-à-dire aucun rapport à la matière de l'expérience, car elles sont bien dotées d'une signification, elles sont même des significations pures, puisqu'elles sont entièrement opérationnelles :

> Dans le sens que prend « signification » dans toute conception ayant une référence existentielle, même indirecte, ses termes n'ont pas de signification — un fait qui explique, probablement, l'idée que l'objet mathématique est simplement une suite de marques arbitraires. Mais dans un sens logique plus large, ils ont une signification, constituée exclusivement et entièrement par leurs relations les uns aux autres, tels qu'ils sont déterminées par la satisfaction de la condition de possibilité d'une transformation[70].

Cette propriété explique le caractère des objets mathématiques d'être eux-mêmes définis relativement à d'autres objets mathématiques, et cela à un niveau de profondeur indéterminé. Cela est possible parce qu'il n'y est jamais question que d'opérations définies relativement à d'autres procédures opératoires, et ainsi de suite. Prenons l'exemple d'une équation différentielle, qui établit une certaine relation entre des fonctions et leurs dérivées. La dérivation désigne une opération que l'on sait faire sur les fonctions réelles, sous certaines conditions. À son tour, une fonction réelle désigne une opération plus ou moins complexe qu'on sait opérer sur les nombres réels. Les nombres réels sont certaines séquences d'entiers, et les entiers sont enfin définis par l'opération élémentaire de comptage. *Tout cela* est présent dans la relation que définit l'équation différentielle et disponible pour la résolution du problème qu'elle peut représenter. Ces signes d'opérations possibles sont comme le plan industriel que Dewey prend pour modèle du jugement[71] ; ils

70. *LTI*, p. 395.
71. Voir plus haut, p. 680.

sont disponibles pour le jugement qui les « mettra au travail » dans la procédure résolvant le problème.

Un point important est donc que toute théorie mathématique, aussi complexes et infinitaires que soient ses objets, est construite à partir d'un nombre fini de règles opératoires à leur propos, qui suivent les trois principes énoncés plus haut, de localité, de composition et d'induction. π n'est pas dénombrable, mais nous savons le manipuler dans des équations algébriques et, lorsque nécessaire, l'approcher avec la précision voulue par une procédure de plus en plus longue, mais toujours finie. On raisonne sur ces opérations, et non sur leurs objets, et c'est pour cela qu'on peut « calculer » à leur propos, c'est-à-dire leur appliquer des procédures systématiques, bien qu'ils soient le plus souvent eux-mêmes infinis[72].

Dans le sens de Dewey, donc, ce sont les mathématiques tout entières qui sont la *science des opérations*, et on peut donc se demander quel rapport elles ont avec le sens beaucoup plus circonscrit que nous avons tantôt donné à ce terme[73]. Nous abordons cette question au chapitre suivant.

§ 177. Le calcul logique et la preuve

Les mathématiques sont considérées à juste titre comme norme de la rigueur scientifique parce qu'elles ne portent que sur des opérations possibles et des transformations possibles d'opérations, par suspension de toute référence aux objets possibles dans l'expérience de ces opérations. Le contrôle de l'inférence peut ainsi être total, puisque celle-ci ne pointe pas vers une connexion déterminée entre des choses de l'expérience. Elles portent sur le raisonnement lui-même, dont elles sont l'expression idéale :

> Le discours ordonné est lui-même une série de transformations conduites selon des règles de substitution de significations rigoureuses (ou nécessaires) et fécondes. Une telle transformation n'est possible que lorsqu'un système de caractères abstraits interdépendants est institué[74].

Cela explique le rôle insigne de la preuve en mathématiques. Dewey la définit comme *production cumulative et procédurale de propositions*[75], qui transforme une formulation initiale du problème en une formulation terminale qui lui soit

72. Voir plus loin, section 11.1.
73. Voir plus haut, section 9.4.
74. *LTI*, p. 392.
75. *LTI*, p. 310.

immédiatement applicable – une action, un objet déjà disponible, etc.[76]. Cette définition, ainsi que l'insistance sur le rôle de la substituabilité dans le progrès de la procédure peut faire penser ici à Aristote[77], ou à la conception hilbertienne de la preuve :

> Une formule sera dite prouvable si elle est un axiome, ou résulte d'un axiome par substitution, ou est la formule finale d'une preuve[78].

Cependant Hilbert parle plutôt de substitutions de formules que de transformation de significations, et cette différence est d'importance. Dewey refuse de séparer la syntaxe et la sémantique, et de concevoir comme logique un simple système de réécritures réglées de formules dont la signification serait mise en suspens. Les significations s'épurent et se transforment, mais ne deviennent jamais un pur système de signes, car elles ne valent que parce qu'elles pointent vers une action possible, et c'est cela qu'est d'abord une signification. Si la logique est symbolique et propositionnelle, ce n'est pas parce qu'elle n'est au fond qu'une grammaire, mais parce que la propriété essentielle du discours tout entier est d'avoir trait à des actions possibles et non à des choses réelles, c'est-à-dire d'examiner ce qui pourrait advenir si on faisait quelque chose, sans pour autant le faire[79].

Dans un système symbolique, les transformations du discours sont réalisées grâce aux opérateurs logiques (les opérateurs ET, OU, etc.) et mathématiques, conçus spécialement pour les rendre rigoureuses (sans indétermination) et productives. Le sens de ces opérateurs est donné par des propositions universelles :

> Les propositions universelles sur les formes logiques sont des fonctions propositionnelles et en tant que telles ne sont en elles-mêmes ni vraies ni fausses. Elles énoncent des modalités de *procédure* dans l'investigation qui sont postulées selon leur applicabilité et leur utilité dans n'importe quelle enquête contrôlée[80].

Si on relève que, pour Dewey, une proposition universelle n'est rien d'autre que l'expression d'une règle opératoire[81], on retrouve ici exactement la perspective de Martin-Löf, à savoir que la signification des constantes logiques réside entièrement dans leurs règles d'introduction et d'utilisation dans les raisonnements. Elles sont donc des fonctions qui permettent de faire progresser l'investigation,

76. *LTI*, p. 313.
77. Voir plus haut, § 107.
78. (Hilbert [1922] 1996, p. 1137).
79. *LTI*, p. 392.
80. Texte original de la deuxième phrase : *They state modes of procedure in inquiry which are postulated as applicable and as required in any controlled inquiry* (*LTI*, p. 159).
81. *LTI*, p. 379.

c'est-à-dire de reformuler ses attentes jusqu'à la ramener à des solutions connues. L'interprétation algébrique des connecteurs logiques permet à Dewey d'approcher cette intuition lorsqu'il interprète la validité de la conjonction en relation avec un acte de preuve :

> La validité de la proposition « James, John, Robert et Henry étaient présents à telle occasion » est bloquée *lorsqu'il est montré* que l'un des quatre était absent[82]. *(Nous soulignons)*

On retrouve ici l'esprit même de l'interprétation intuitionniste des constantes logiques de la négation et de la conjonction. L'assertion de la négation d'un jugement consiste à montrer l'impossibilité d'une preuve, et l'assertion d'une conjonction requiert d'exhiber les preuves de chacun de ses termes. Un autre exemple montre que Dewey avait assez clairement perçu ce qui correspond aux règles d'introduction dans la théorie constructive des types :

> Prenons, par exemple, un postulat tel que le suivant : « Si a et b sont des éléments du corps K, alors ab $(a \times b)$ sont des éléments de K ». Les éléments postulés sont a, b. Les opérations postulées sont représentées par « et » et par « $\times$ » ou ab. La proposition primitive ne postule pas d'abord certains éléments, puis, au moyen d'une autre proposition primitive, postule une certaine opération sur deux postulats séparés. Les éléments et les opérations sont énoncés dans un seul postulat, et sont en dépendance logique les uns des autres. a est défini de telle sorte que, si l'opération désignée par *et* est applicable, alors l'opération symbolisée par $\times$ est nécessairement applicable. Les éléments sont institués par rapport aux opérations qui les mettent en rapport et les opérations et leurs règles sont déterminées par référence aux éléments [83].

Cette dernière idée, que les termes et les opérations d'une algèbre sont co-définis exclusivement l'un par l'autre est bien connue depuis les algébristes anglais. Dewey tente ici de prévenir l'interprétation que ce postulat décrirait des structures relationnelles abstraites, autrement dit, pour utiliser les termes consacrés aujourd'hui, que sa signification résiderait dans les modèles qui le satisfont. Il insiste au contraire sur le fait qu'il s'agit de définitions stipulatives (cf. les termes *postulés, institués*) qui autorisent l'emploi d'une opération si ses éléments satisfont certaines conditions.

82. *LTI*, p. 337.
83. *LTI*, p. 403.

§ 178. Dewey et l'intuitionnisme

On peut reconnaître dans cet exposé plusieurs caractéristiques de la logique intuitionniste relevées plus haut (§ 148). D'abord, pour Dewey, il n'y a pas de distinction fondamentale entre théorème et problème, et la preuve d'un jugement est donc une procédure pour les mêmes raisons qu'évoque Martin-Löf : elle est une construction qui exhibe son objet, la proposition. Ensuite, le jugement, comme acte de connaissance, doit être compris comme le couple constitué par l'objet *et* sa procédure de construction ou encore par la proposition *et* sa preuve. Enfin, il n'y a pas de distinction stricte entre objets et propriétés, (les individus et les prédicats de la logique du premier ordre), car toutes sont des significations pointant vers des expériences vécues et des opérations possibles dans l'expérience. On peut ajouter à cela que la vérité est définie comme *assertabilité garantie* (*warranted assertability*) une expression célèbre de Dewey que les intuitionnistes ont souvent repris pour désigner leur propre conception de la vérité[84].

Ces rapprochements pourraient être prolongés sur d'autres thèmes, par exemple concernant le refus du formalisme, c'est-à-dire de la séparation de la syntaxe et de la sémantique[85], le rejet du principe du tiers-exclu, ou la définition des nombres et de l'infini[86]. Quel est donc le sens de ces rapprochements possibles entre Dewey et Martin-Löf, ici pris comme représentant emblématique de l'école constructiviste en logique ? Rien n'indique que Dewey ait lu Brouwer ou Heyting, ou même ait entendu parler de leurs travaux. Cependant, il cite Poincaré à propos du principe d'induction, qu'il accepte comme ce dernier de considérer comme indérivable d'autres principes, mais qu'il refuse de justifier par un appel à l'intuition[87]. On peut considérer que cette critique est bénigne, et qu'elle suppose un accord de fond sur la conception des nombres et du raisonnement mathématique qu'expose Poincaré. Dewey n'était certainement pas hostile au scientifique français, et on peut même avancer que sa lecture a orienté sa conception des mathématiques puisqu'il se réfère plusieurs fois à ses ouvrages dans d'autres textes, et qu'il le place parmi les savants dont les réflexions sur « la portée logique des modes de procédure » de sa science[88] ont le plus fait avancer

84. (Posy 1998, p. 303).
85. Voir à ce sujet Sundholm (1997, p. 191-192).
86. Voir, respectivement, *LTI*, p. 344-345, 362, 409.
87. *LTI*, p. 405.
88. MW 3 : 68. Voir également MW 4 : 262.

la logique contemporaine.

Au-delà de cette influence de Poincaré, le sens de ces rapprochements vient surtout du fait, qu'adoptant la même perspective fondamentale sur la nature de la pensée comme activité, Dewey et Martin-Löf parviennent aux mêmes résultats, ou en tous les cas à des thèses qui se font signe. Ils affirment ainsi tous deux l'unité de la connaissance théorique et de la connaissance pratique, sous l'égide de la seconde, une fois que celle-ci est bien comprise comme *savoir de la résolution de problème*. Les jugements sont les actes complexes qui effectuent cette résolution, et ils ont un primat logique sur leur résultat, c'est-à-dire la proposition que cet acte construit. Comme Dewey surtout, Martin-Löf ne réduit pas la portée de la logique aux seuls actes de connaissance, mais la considère être immanente à l'activité rationnelle en général, parce qu'elle ne fait rien d'autre que décrire les conséquences de ce qu'on prétend faire[89]. Cependant Martin-Löf ne développe pas une philosophie de l'action, et ses indications pour prolonger la théorie des types en direction des sciences naturelles est trop parcellaire. De son côté Dewey, qui reste techniquement tributaire de la logique algébrique de Boole, et du calcul simple des propositions, n'a pas accès au calcul des prédicats du premier ordre et à la quantification. Il ne peut donc parvenir aux opérateurs fonctionnels (abstraction et application) du λ-calcul, ni donc à une vision générale et contemporaine de la calculabilité.

Faut-il donc comprendre de ces rapprochements que la théorie des types, ou une logique constructiviste équivalente, aurait la capacité d'exprimer, en termes formels, le contenu de la *théorie de l'investigation* que développe Dewey ? Cette question, posée ainsi, ne nous semble pas avoir de sens, puisque Dewey assigne une place très précise aux formes symboliques dans l'investigation. L'investigation peut développer des systèmes de symboles pour représenter des opérations possibles, mentales ou concrètes, ainsi que leurs enchaînements potentiels. Mais rechercher un système symbolique qui représenterait ces opérations de l'investigation développant des systèmes reviendrait à vouloir, comme le baron de Münchhausen embourbé dans une mare, en sortir en se tirant soi-même par les cheveux. C'est en tous les cas l'impression que donne la curieuse tentative de Burke (2002) de trouver des correspondances aux idées de Dewey dans différentes logiques formelles contemporaines, par exemple lorsqu'il propose de représenter les *théories* dont dispose un agent par une structure formelle d'ensembles, qui repré-

89. Voir plus haut, p. 635.

senteraient, entre autres, un réseau de significations pré-théoriques, une collection d'ontologies et de structures de raisonnements, etc.[90]

Il faut plutôt inverser la perspective : *c'est l'investigation qui, une fois qu'elle a atteint son terme, se réexprime elle-même*, dans les systèmes symboliques qu'elle a développé, *en une procédure qui la récapitule*. La procédure n'est pas toujours explicitée, et la plupart de nos investigations sont aussitôt oubliées qu'elles se sont achevées avec satisfaction. Mais on juge parfois utile d'en consigner certaines, afin de contrôler que la suite d'inférences effectuées est correcte, ou pour la partager avec autrui qui la vérifiera, ou pour se la rappeler afin de ne pas avoir à réitérer l'investigation dans une situation similaire, ou pour l'enseigner, etc. Telle est la fonction des preuves, mais également par exemple des procédures calculatoires, ou encore des scriptings d'actions collectives complexes.

Une telle démarche n'est pas associée à un système symbolique spécifique. Elle peut prendre la forme de la géométrie, de l'arithmétique, du calcul des propositions, de la théorie des graphes, etc. L'ambition fondationnelle spécifique de la théorie constructive des types, qui la distingue par exemple de la théorie des ensembles, mais qui le rapproche du λ-calcul ou des systèmes de Post, est de prétendre pouvoir exprimer *toute procédure*, ce qui l'oblige également à pouvoir représenter tout système symbolique qui décrit son champ de significations, et dans lequel elle s'exprime comme une composition d'activités et de spécifications. On peut mesurer l'ampleur de cette ambition aux difficultés déjà évoquées que la théorie des types rencontre à représenter l'axiomatique euclidienne[91].

L'idée qu'il *doit* être possible de représenter toute procédure effective (nous revenons sur cette notion au chapitre suivant) dans un seul système symbolique est devenue intuitive pour nous depuis les années 1930 et les travaux de Church et Turing. C'est à ce titre que la théorie des types, ou d'autres systèmes équivalents, peuvent prétendre à une relation spéciale avec la théorie de l'investigation telle que l'expose Dewey. Il ne faut pas interpréter ce cadre comme celui que doit *utiliser* en pratique le mathématicien pour rechercher des preuves et des solutions, même s'il peut lui être utile de le faire dans certains cas, ni même comme celui dans lequel il *doit* les exprimer une fois qu'il les a trouvées afin de s'assurer qu'elles sont correctes – car il peut y avoir d'autres systèmes plus commodes de vérification pour son domaine – mais comme celui dans lequel il *peut* toujours les exprimer, quel

90. (Burke 2002, p. 137).
91. Voir plus haut, section 9.2.

que soit son sujet. La théorie des types prétend être une forme idéale et universelle de la notion de procédure, un cadre dans lequel toute procédure peut être l'objet d'une explicitation exacte et exhaustive, celle que doit produire le mathématicien si on lui demande d'effectuer dans son entièreté le jugement de connaissance qu'il affirme *pouvoir* effectuer.

C'est donc uniquement par ce long détour que la notion de procédure, telle que la dégage Dewey, peut être effectivement connectée à notre notion moderne de procédure informatique.

Dans la conception de Dewey se trouve un progrès épistémologique essentiel, qui évite l' « oubli » de la délibération que nous avions trouvé chez Kant aussi bien que chez Martin-Löf et Sundholm[92]. La théorie de l'investigation de Dewey permet de distinguer clairement entre l'investigation et la procédure, entre le processus complexe et tortueux de la première et l'exécution nette et mécanique de la seconde. Ce progrès est décisif car *cet écart* est la raison de la dynamique cumulative du savoir qui, dans chaque jugement ou acte de connaissance, est capable de se récapituler en une procédure compacte, capable éventuellement de devenir une habitude, ou d'être signifiée comme une opération simple manipulable à son tour – de devenir un objet.

Ainsi, le paradoxe que nous avions relevé au chapitre précédent, que l'acte idéal de la preuve est justement celui que *jamais* un mathématicien ne daigne réaliser dans son exhaustivité laborieuse[93], a une réponse très simple dans la perspective de Dewey. L'investigation vise à dégager cet acte idéal, mais elle ne se confond certainement pas avec lui. La procédure représente l'investigation une fois celle-ci achevée, et une fois éliminées toutes les erreurs qu'elle a pu commettre en chemin et les voies sans issue qu'elle a explorées, car sa fonction est d'aider les investigations futures à se former en « balisant » le bon chemin, à permettre sa vérification, voire à la réduire à une pure ratiocination. Cette séparation fonctionnelle est très claire dans le cas spécifique où l'investigation mathématique a pour rôle de déterminer les méthodes de calcul optimales pour un problème donné, afin qu'elles soient exécutées par des *calculateurs* – des personnes humaines, avant l'avènement des ordinateurs, comme les fameux « perruquiers » que Prony avait embauchés pour calculer les tables de logarithmes sous la direction de mathématiciens[94]. On

92. Voir plus haut, § 142, et § 163.
93. Voir plus haut, p. 639.
94. Voir plus haut, § 85.

ne demande surtout pas à ces calculateurs laborieux de mener l'investigation à nouveau, mais au contraire d'appliquer à la lettre les instructions données par des mathématiciens. Il s'agit ici d'une véritable division du travail, selon l'aveu même de Prony qui dit avoir eu son idée après avoir lu Adam Smith.

10.4 Système et théorie

Nous sommes à présent en mesure d'aborder la question, cruciale pour notre réflexion, de la nature des systèmes techniques. Dans cette section, nous prenons pour modèle le rôle que jouent les systèmes scientifiques (*i.e.* les théories) dans la résolution de problème, avant de raisonner par analogie, à la section suivante, à propos des systèmes techniques. Nous étudions d'abord un texte de jeunesse de Dewey, qui en donne l'idée générale (§ 179). Nous montrons ensuite que l'idée de *théorie* est remplacée dans les textes de la maturité par l'expression plus générique de *système de significations*, et qui désigne le contexte d'arrière-plan de l'investigation. La démarche scientifique consiste à établir des systèmes de signification contrôlés (§ 180). De là découlent son exigence de systématicité (§ 181) et sa dynamique cumulative remarquable (§ 182).

Il faut noter ici que, en exposant avec approbation la doctrine de Dewey concernant le rôle des théories scientifiques dans la résolution de problème et la dynamique de leur formation, nous n'affirmons pas là qu'en cela se résumerait toute la « vérité » de la science. Nous n'excluons pas que les théories scientifiques aient d'autres fonctions et d'autres dynamiques ; nous affirmons seulement qu'elles ont *au moins* celles que nous exposons dans cette section. Cette remarque ne fait que reprendre d'autres que nous avons déjà faites concernant le minimalisme épistémologique auquel nous tentons de nous tenir[95].

§ 179. La théorie de la situation

Un texte de jeunesse exprime l'idée générale de Dewey par la formule lapidaire qu'*il y a théorie dès qu'il y a action* :

> Dès […] qu'il y a une chose qui mérite le nom de conduite, il y a une idée, une « théorie », au moins aussi large que l'action. Ce n'est pas parce que la théorie a une portée étroite qu'elle n'est pas présente ; et elle n'est étroite que dans la mesure

95. Voir plus haut, section 1.5, et également § 144.

> où l'acte correspondant est abstrait et partiel. L'homme moyen peut marcher sans trop de théorie, parce que marcher n'est pas un acte de grand contenu […] Pour tout acte (par opposition à une simple impulsion), il doit y avoir une « théorie », et plus l'acte est large, plus son importance est grande, plus la demande de théorie est exigeante[96].

Cette thèse est affirmée dans le contexte d'une discussion sur l'existence et la nature de théories morales. Dewey soutient qu'elles ne peuvent être une somme de principes abstraits, mais que chaque situation appelle à une analyse de ce qu'il faut faire, en tenant compte de toutes les particularités, contraintes et objectifs en jeu. Les règles morales sont des outils pour guider la clairvoyance morale (*moral insight*) mais ne peuvent déterminer l'action, car elles sont, comme toutes les règles, abstraites ou hypothétiques. C'est seulement dans la réflexion morale située que peut apparaître une idée intégrée et concrète de ce qu'il faut faire. Sans doute une telle analyse sera incomplète, mais on vise toujours une telle idée, et c'est cela qu'il faut d'abord appeler théorie, la *théorie de la situation*, l'idée qui guide l'action. Lorsqu'on apprend à marcher ou à compter, un effort intellectuel est fourni pour comprendre comment diriger notre corps et notre esprit. On construit ainsi une théorie de ce qui fonctionne et ce qui ne fonctionne pas, même si elle n'est pas articulée, et qu'elle est directement incorporée à l'habitude. Ces théories n'apparaissent pas comme morales car elles visent à obtenir une compétence générale, mais elles le sont en fait, car ces compétences sont des biens.

L'influence de Hegel est encore prégnante dans cette analyse, par exemple dans l'idée que l'action morale doit intégrer toutes les contraintes pratiques et autres déterminations de la situation – par opposition à l'abstraction de la loi morale kantienne, ou encore que le concret est un surcroît, et non un défaut, de rationalité. Dans cette perspective, Dewey passe donc allègrement outre le fait qu'on appelle traditionnellement théories des systèmes discursifs de représentations, et que bon nombre d'actions s'en passent entièrement.

Cependant là n'est pas l'essentiel, et si on accepte de prendre *théorie* dans le sens élargi que Dewey lui donne ici, cette thèse découle directement, à plusieurs niveaux, des exposés faits ci-dessus. L'idée essentielle est que toute action, et même tout comportement, se déploie sur un arrière-plan de significations, d'inférences, ou d'habitudes déjà acquises, qui déterminent à la fois sa visée et ses moyens. Toute résolution rationnelle à agir requiert un réseau de significations dans lequel

96. EW 3 : 96.

l'investigation puise ses moyens matériels et procéduraux. Dans le cas d'une réflexion rapide et intuitive, on se fonde sur les suggestions qui se présentent dans l'appréhension immédiate du problème. Dans le cas d'une conduite habituelle ou routinière il y a, peut-on dire, comme un pré-cadrage de l'expérience – ainsi, si je marche, que le sol est ferme devant moi. Et en poussant ce registre métaphorique à son terme, on pourrait même aller jusqu'à formuler que la bactérie qui suit un gradient ionisé dans un milieu aqueux « a la théorie » qu'un tel gradient mène à la nourriture. Bien entendu, l'idée principale à travers tous ces exemples n'est pas qu'il y ait raisonnement, représentation, ni même sentience, mais que l'action, essentiellement générique, projette une lecture également générique du milieu dans lequel elle se déroule – qu'elle s'adapte aux traits seuls qu'elle sait y reconnaître.

L'intérêt de ces extrapolations métaphoriques est, qu'une fois retournées, elles montrent que les théories scientifiques sont la forme la plus élevée d'un arrière-plan de significations ou d'habitudes qui a toujours été là, et qui est nécessaire à toute action et investigation, rationnelle ou non, discursive ou non, parce que celles-ci vont y puiser leurs moyens et leurs matériaux. Les théories scientifiques visent à réformer les inférences naturelles inscrites dans le langage, mais le langage lui-même se construit sur les pratiques et les interactions d'une communauté, et celles-ci à leur tour sont des adaptations collectives aux pressions et aux opportunités d'un milieu organique donné.

Cependant, le défaut de l'exposé de 1891 est que, s'il met l'accent sur la complémentarité nécessaire entre théorie et pratique, il ne permet pas de rendre compte de cette dynamique de progrès que met en branle la compétence réflexive de la personne, capable de porter au langage ses propres pratiques, et donc de les observer de manière critique, et même d'observer son propre langage afin de contrôler sa pertinence et sa fiabilité. Si tout est théorie, comme expliquer la spécificité et la supériorité des théories scientifiques ?

C'est pour cette raison sans doute que Dewey, contrairement à son usage très large du terme *théorie* en 1891, le restreint à l'extrême dans la *Logique*, en l'appariant quasi-exclusivement à l'adjectif *logique* (*logical*). L'expression *théorie scientifique* n'y apparaît qu'à un endroit[97], dans le cadre d'une discussion historique. On trouve aussi des mentions péjoratives de théories politiques ou psychologiques en fin d'ouvrage, et plus généralement, un emploi non technique du terme, où il est pris comme une expansion des notions d'idée, d'hypothèse, de loi scientifique.

97. *LTI*, p. 70-71.

Cette omission est certainement voulue. Dewey précise méticuleusement ce qu'il entend par investigation ou méthode scientifique, mais il combat l'identification de la science à des corpus de propositions générales (lois, théorèmes) qu'on associe aujourd'hui au terme *théorie*. Une telle « restriction dogmatique de la science[98] » établit en effet une scission artificielle au sein des nombreuses démarches scientifiques, rejetant hors de la « science » non seulement les techniques, mais également les activités expérimentales des sciences elles-mêmes.

Cette méfiance à l'égard de ce terme provient de ce que Dewey perçoit ce qui persiste en lui de l'ancienne *θεωρία*, sa dimension contemplative qui établit un face à face intenable entre le sujet et l'objet, et qui entraîne ce qu'il dénomme de manière célèbre la *théorie de la connaissance-spectacle* (*spectator theory of knowledge*) :

> La théorie de la connaissance est calquée sur ce qui est censé se produire dans l'acte de vision. L'objet qui réfracte la lumière vers l'œil est vu ; cela fait une différence pour l'œil et pour celui qui dispose d'un appareil optique, mais cela n'en fait aucune pour la chose vue. L'objet réel, qui se tient fixe dans sa distance majestueuse, est un roi pour tout esprit contemplatif qui pourrait lever les yeux sur lui. Une théorie de la connaissance-spectacle en est le résultat inévitable. Certaines théories ont soutenu que l'activité mentale intervenait, mais elles ont conservé l'ancienne prémisse, et en ont donc conclu qu'il est impossible de connaître la réalité. Si l'esprit intervient, nous ne connaissons, selon eux, qu'une ressemblance modifiée de l'objet réel, une « apparence ». […] Toutes ces notions de certitude et de fixité, de nature du monde réel, de nature de l'esprit et ses organes de la connaissance, sont complètement liées les unes aux autres, et leurs conséquences se ramifient dans pratiquement toutes les idées importantes entretenues dans n'importe quelle question philosophique. Elles découlent toutes — telle est ma thèse élémentaire — de la séparation (établie dans l'intérêt de la recherche de la certitude absolue) entre théorie et pratique, entre savoir et action[99].

§ 180. Les systèmes de significations

Cependant, l'idée d'un arrière-plan de significations nécessaire au développement de l'investigation est toujours là, et Dewey emploie pour la désigner l'expression *système de significations* (*system of meanings*) qui est très présente dans *Expérience et Nature* et dans la *Logique*, et se trouve régulièrement à travers ses autres écrits. Dans *Expérience et nature*, c'est l'intellect lui-même (*mind*) qui est qualifié

98. *LTI*, p. 434.
99. LW 4, 19-20.

tel[100]. Dans la *Logique* de 1938, elle est associée au langage qui unit une communauté humaine donnée[101]. Au-delà de ces matrices primitives de la connaissance, des systèmes de significations spécialisés sont institués par les personnes humaines – par exemple des codes ou des rites spécifiques à un sous-groupe, des langages spécialisés ou encore des théories scientifiques, qui visent à réformer les significations de manière à mieux contrôler les inférences. L'établissement d'un « système de significations cohérent et auto-éclairant (*self-illuminating*) » est la visée de la science[102].

Ce qui transparaît à travers tous ces usages de l'expression par Dewey est qu'elle s'oppose à l'investigation et à l'action comme, chez Aristote, la puissance à l'acte. Qu'il s'agisse de langage, d'esprit, de codes, de sciences – toutes ces dénominations ne recouvrent aucun objet phénoménal concret en particulier, mais désignent ce qui est à disposition de l'individu à travers l'habitude, les pratiques collectives, les dispositifs matériels comme les livres ou les bibliothèques. Cependant, contrairement peut-être à l'idée aristotélicienne, la puissance ici n'est pas passive, mais une dynamique qui oriente et suscite l'action, qui sculpte l'expérience. On peut se référer ici à l'analyse par Hildebrand (2008, p. 24-25) du concept d'habitude chez Dewey. Le passage où Dewey qualifie l'intellect de *système de significations* est sans doute le plus éclairant à cet égard :

> La relation entre l'intellect (*mind*) et la conscience peut être indiquée par un événement familier. Lorsque nous lisons un livre, nous sommes immédiatement conscients des significations qui se présentent et s'évanouissent. […] Mais nous sommes capables d'extraire des idées de ce qui est lu grâce à un système organisé de significations dont nous ne sommes à aucun moment complètement conscients. Notre « intellect » mathématique ou politique est le système des significations qui possèdent et déterminent nos appréhensions ou idées particulières [dans ces domaines]. Il y a cependant un continuum ou un spectre entre ce système englobant et les significations qui, étant focales et urgentes, sont les idées du moment. Il y a un champ contextuel entre ces dernières et les significations qui déterminent la direction habituelle de nos pensées conscientes et fournissent les organes de leur formation. Une grande erreur de la tradition psychologique orthodoxe est sa préoccupation exclusive sur la focalisation nette [de la conscience] au détriment de l'obscurité et du vague qui gagnent à mesure qu'on s'en éloigne vers un champ d'une clarté décroissante[103].

100. LW 1, 230-231.
101. *LTI*, p. 56.
102. MW 2 : 307.
103. LW 1, 231.

Se dégage ici assez clairement l'idée d'une puissance active, située à l'orée de la conscience, entourant et pressant cette clairière ; cette description est typique du style de pensée de Dewey, qui évite les oppositions catégorielles franches, et cherche toujours à mettre en avant la dynamique de l'expérience et de la pensée.

La science s'inscrit dans la démarche très large de la réflexion pour *prendre contrôle* de cet arrière-plan de significations qui exerce sa puissance sur elle, en l'explicitant et en le soumettant à un examen critique.

Les trois exemples d'investigation donnés plus haut sont utiles pour illustrer la gradation critique qui amène à la science. Celle qui cherche à établir un plan d'action peut être dite critique en un sens figuré, en ce qu'elle suspend l'action irréfléchie pour faire intervenir la réflexion. Mais à part cela, elle est purement constructive, en ce qu'elle assemble en une solution des matériaux inférentiels qu'elle trouve et prend dans la situation de manière non critique ; elle agit dans le système du sens commun. Dans le cas du procès, au contraire, il s'agit de vérifier et de valider une seule inférence complexe, qui part du matériau des faits disponibles et qui conduit vers la culpabilité ou l'innocence du prévenu. Le procès s'appuie sur un système de significations, le droit, soigneusement établi et contrôlé. Enfin, dans le cas scientifique, il s'agit de déterminer des faits ou des lois qui ont vocation à être insérés dans un tel système de significations contrôlé. On est ici dans l'investigation théorique à proprement parler. Il existe un quatrième niveau critique d'investigation : la logique, comme méta-théorie (on a vu que Dewey parle simplement ici de *théorie*, et réserve ce terme à ce niveau ultime) est une investigation sur l'investigation, et vise donc à dégager l'ensemble de ces mouvements critiques, ainsi que leurs schémas universels – leurs principes directeurs (*leading principles*), selon l'expression que Dewey emprunte à Peirce.

La gradation qui vient d'être esquissée a une vocation suggestive, et ne doit pas être comprise comme une classification rigide. Certains jugements du sens commun adoptent le sérieux et la méticulosité du tribunal. Des décisions judiciaires peuvent prendre valeur de jurisprudence et venir ainsi augmenter le système du droit. Il est parfois difficile de distinguer les problèmes scientifiques appliqués et ceux qui sont destinés à affermir le système de la science. Et ce sont souvent des réflexions sur la méthode de la science – des réflexions logiques au sens de Dewey – qui permettent à celle-ci de progresser. Par ailleurs, des investigations complexes peuvent devenir habituelles, voire entrer dans le sens commun, si elles sont d'un usage courant.

« Les entreprises d'une époque deviennent le sport et le loisir d'une autre[104]. »

On voit cependant à travers ces exemples que la possibilité est donnée à la réflexion d'opérer des retours critiques, non seulement sur ses propres productions, mais également sur les systèmes de significations qui la sous-tendent. Il ne peut être question de développer ici toutes les implications épistémologiques de ce fait, ainsi que ces différentes modalités de retours critiques que nous avons esquissées. Nous allons simplement tenter de caractériser la nature de cette *critique des systèmes* qu'est la science, ainsi que sa dynamique cumulative.

§ 181. La décontextualisation et la systématicité

On peut aborder le premier point par la formule que la science critique les systèmes de signification *en ce qu'ils ne sont pas assez systématiques*. Dewey revient tout au long de sa *Logique* sur cette exigence de systématicité. À ce titre, le langage naturel en a un tel défaut que Dewey lui substitue parfois l'expression suggestive d'une *constellation de significations* :

> Les coutumes, l'ethos et l'esprit d'un groupe sont le facteur décisif dans la détermination du système de significations en usage. Le système peut être dit « un » en un sens pratique et institutionnel plutôt qu'intellectuel. Les significations formées sur cette base contiennent certainement de nombreuses choses non pertinentes et excluent de nombreuses autres nécessaires à un contrôle intelligent de l'action. Les significations sont grossières et beaucoup d'entre elles sont incompatibles les unes avec les autres d'un point de vue logique. Une signification est appropriée à l'action sous certaines conditions institutionnelles, une autre l'est dans une autre situation, et il n'y a aucune tentative de relier les différentes situations les unes aux autres dans un schéma cohérent. D'un point de vue intellectuel, il y a plusieurs langages, bien que du point de vue social il n'y en ait qu'un. Cette multiplicité de constellations de significations linguistiques est aussi une marque de notre culture existante. Un mot signifie une chose par rapport à une institution religieuse, encore une autre chose dans les affaires, une troisième chose dans le droit, et ainsi de suite. Ce fait est la véritable tour de Babel de la communication[105].

Le langage n'est donc pas vraiment un système, mais une simple constellation de significations, en ce qu'il permet d'exprimer, de manière disparate et ambiguë, toute forme de pratiques et d'opérations issues de groupes sociaux différentes, car sa fonction est justement de permettre leur communication, de faire dialoguer

104. *LTI*, p. 70.
105. *LTI*, p. 56.

l'homme d'affaires, le prêtre et l'artisan, à propos de choses dont ils considèrent chacun des potentialités et des conséquences différentes. De ce fait, le langage accroît un défaut qu'ont toutes nos inférences, même au sein d'un contexte homogène : c'est qu'elles sont justement adaptées à ce contexte, et qu'elles ne disent rien de leur propre validité sous de nouvelles conditions et circonstances ; la pratique a une certaine tolérance pour l'exception [106].

Le rôle des théories scientifiques est de s'assurer que les inférences sur lesquelles s'appuient nos actions sont valides, c'est-à-dire réussies, ce qui les amènent à explorer systématiquement leurs conditions et leurs conséquences, et ainsi à les reformuler radicalement :

> Chaque signification qui entre dans le langage [scientifique] est déterminée expressément dans sa relation aux autres membres du système de langage. En tout raisonnement ou discours ordonné, ce critère a priorité sur celui institué par sa connexion avec les habitudes culturelles [...] L'idéal d'un langage scientifique est la construction d'un système dans lequel les significations sont reliées l'une à l'autre par l'inférence et le discours et où les symboles sont tels qu'ils indiquent cette relation[107].

On voit ici que la systématicité signifie le *contrôle* du rapport de chaque signification avec les autres membres du système, et que ce rapport est soit établi par l'inférence, soit par le discours. On reconnaît ici la dualité entre moyens matériels et moyens procéduraux, qui permet de caractériser ce que veut dire *systématiser* par un double travail de contrôle. Il s'agit d'une part d'*ordonner* les inférences empiriques les unes par rapport aux autres, et d'autre part d'expliciter la nécessité interne de cet ordre, c'est-à-dire de la ramener à un ordre logique. Tout cela ne dit pas encore grand-chose d'original, si l'on ne se rappelle pas que les significations traitent des *opérations* possibles dans l'expérience et de leurs conséquences (qui pointent vers d'autres opérations). La première partie de la systématisation d'une théorie consiste donc à déterminer quelles opérations sont primitives ou prioritaires dans l'établissement des faits, et la seconde, à montrer comment (c'est-à-dire par quelles règles) toutes les autres peuvent être *générées* ou examinées à partir de celles-ci.

Bien entendu, au cours de ces travaux, on s'aperçoit que les opérations initialement candidates à être le socle du système souffrent des exceptions, ou que leur relation à d'autres opérations est plus complexe qu'anticipée, et elles sont donc corrigées ou rejetées en conséquence. Cela explique d'une part l'abstraction crois-

106. MW 1:23.
107. *LTI*, p. 56.

sante de la science, qui semble définir des termes théoriques sans rapport apparent avec l'expérience immédiate, comme « *masse* » ou « *force* », et d'autre part sa mathématisation, car elle n'exprime plus que des rapports entre ces opérations, et non plus des inférences immédiatement réalisables. Dewey donne plusieurs exemples de ces travaux tout au long de la *Logique*. Nous en choisissons deux ici.

Concernant l'établissement des faits primitifs, on peut prendre l'exemple de l'évolution de notre conception des métaux[108]. La malléabilité d'un métal était signifiante dans la conception ancienne, elle permettait de reconnaître un matériau comme métal mais également de discriminer différents métaux entre eux par leur plus ou moins grande torsion à une pression équivalente. Cette propriété était issue directement de l'expérience pratique que l'on avait des métaux, comme matériaux qu'on pouvait former à des fins et à des utilités diverses. Dans la conception moderne, ce type de propriété n'est plus essentiel. La description des métaux se fait à l'aide de formules chimiques qui permettent de déduire directement les résultats d'un grand nombre d'opérations possibles sur ce corps, y compris les opérations liées à la malléabilité, mais également sa réaction avec des composés chimiques, comme l'oxygène ou le soufre, ou encore la conductivité électrique. On dispose ainsi d'un plus grand nombre de manières de reconnaître les métaux, ce qui permet d'en découvrir de nouveaux, d'en distinguer les différents isotopes, ou de les décrire bien plus précisément que par les anciennes qualités immédiates. Mais d'autre part, on a aussi grandement augmenté par là les possibilités qui s'ouvrent à nous dans leurs usages, et l'analyse des phénomènes les concernant.

Les formules chimiques n'ont de signification qu'au sein d'une théorie donnée, la chimie, qui permet d'expliciter toutes leurs conséquences. Elle dit, par exemple, comment calculer la conduction et la densité d'un métal à partir de sa formule, ou sa réactivité à l'oxygène, et établit donc des liaisons nécessaires entre ces opérations. La formule chimique est ainsi une signification qui se trouve à l'intersection de toutes les opérations réalisables sur les éléments et permet de les mettre en relation réglée, et non plus seulement empirique. La dernière étape de la systématisation consiste à disposer d'une définition *disjonctive* des métaux qui permet de les identifier tous *a priori*. Cette définition est donnée par la table périodique des éléments, où le groupe des métaux est distingué par le faible delta d'énergie séparant la bande de valence et celle de conduction ; on s'aperçoit d'ailleurs, à l'occasion de cette définition systématique, qu'il existe des éléments semi-métalliques, c'est-

108. *LTI*, p. 274.

à-dire qui peuvent se comporter comme des métaux sous certaines conditions, ce qui illustre parfaitement la position de Dewey. On retrouve bien ici la définition kantienne du système comme « unité des diverses connaissances sous une idée[109] », qui apparaît bien dans sa fonction régulatrice et critique des concepts naturels. Il ne s'agit pas seulement de réorganiser le savoir selon des visées esthétiques, mais bien de le reconstruire.

Un autre exemple que prend Dewey[110] pour illustrer cette réduction logique des liaisons entre différentes opérations possibles, est la proposition *Tout ce qui est rare est cher*. On peut interpréter cette proposition de deux manières. Soit elle est empirique : elle affirme que, dans le tout vaguement délimité de l'expérience – par exemple dans l'économie française d'aujourd'hui, ce principe a été vérifié et continuera de l'être. Soit elle signifie qu'il existe une liaison nécessaire entre la difficulté qu'il y a à trouver une chose dans le commerce et le prix qu'il la faut payer, du fait, par exemple, de la concurrence que se font les acheteurs pour l'obtenir. Dans le second cas, on établit une proposition universelle, qui vaut même s'il n'y a plus de rareté des biens. Il n'y a pas besoin de vérifier une telle proposition dans l'expérience, et si celle-ci l'infirme, c'est qu'on parlait d'autre chose[111]. On peut en un certain sens dire que ce lien est analytique, mais cela ne veut pas dire qu'on peut déduire par syllogisme la cherté de la définition de la rareté. Dewey d'ailleurs n'utilise pas cet adjectif kantien. Il insiste par contre sur l'*analyse* de ce qu'impliquent les opérations en jeu – s'approvisionner, négocier, fixer un prix, etc. qui permet d'établir un ensemble de définitions coordonnées dans un système de significations où la loi en question peut alors apparaître comme un théorème. Même si Dewey ne prolonge pas cet exemple, on peut examiner comment Walras ([1874] 1988) traite la question : il construit un modèle mathématique de l'équilibre de marché, où l'achat et la vente se déroulent selon un mécanisme d'enchères qui permet de montrer la relation explicite entre la fixation du prix et la tension de l'offre et de la demande. On voit bien ici que dans la démarche scientifique d'investigation, la définition est un processus ardu, qui non seulement peut exiger de réaliser des distinctions de sens dans des termes initialement communs, de considérer comme accessoire ce qu'on pensait être essentiel, etc. mais aussi, parfois, de reconfigurer entièrement un système de significations et de le porter sur

109. Voir plus haut, p. 545.
110. *LTI*, p. 304.
111. *LTI*, p. 301.

un plan purement mathématique afin d'assurer la robustesse des inférences qui en découlent.

Le modèle de Walras est en quelque sorte une fiction, car jamais les prix ne sont fixés par le modèle décrit des enchères. Il a le même statut que le principe d'inertie qui décrit le mouvement d'un corps isolé, qu'on ne trouvera jamais dans la nature. Il ne s'agit pas là d'abstractions ou d'idéalisations, car l'intérêt n'est pas de décrire des situations qui seraient proches de ces cas de figure idéaux, mais des principes opérationnels qui peuvent guider l'établissement d'inférences entre différentes opérations de mesure[112], comme en géométrie.

Un autre enseignement de cet exemple est de bien montrer en quel sens nous pouvons affirmer que l'attitude théorique consiste à « décontextualiser » le savoir. En s'attachant seulement à contrôler la validité d'une inférence, l'investigation théorique doit expliciter les différents contextes dans lesquels elle peut survenir, c'est-à-dire les problèmes qui la mobilisent – avec leurs finalités, leurs présupposés, leurs contraintes – et ainsi à la reformuler sous une forme qui convient à tous, et qui ne dépend donc d'aucun, quitte à la spécifier voire à l'éclater dans un nouveau réseau d'inférences plus spécialisé. L'assertion *« Tout ce qui est rare est cher »* est énoncée par des hommes d'affaires ou des consommateurs qui s'inquiètent de leurs intérêts, par exemple à propos du pain une année de sécheresse, d'un trafic de diamants, ou encore d'un voyage vers la Lune. Ces situations ont des dynamiques de formation des prix très différentes et les stratégies de leurs protagonistes sont incomparables. Cependant la valeur du modèle de l'équilibre général des prix est de convenir à toutes ces situations, en se désintéressant des comportements réels des acteurs mais en en montrant la logique sous-jacente. En rendant ainsi le savoir disponible pour toute résolution de problème, l'attitude théorique d'investigation vise à ne dépendre d'aucune.

Organiser ainsi les problèmes, montrer leur identité à travers la diversité apparente de leurs conditions, ou établir leur dépendance les uns vis à vis des autres de manière à factoriser les problèmes les plus élémentaires, est une démarche qui conduit à l'attitude systématique telle que nous l'avons caractérisée par les principes de composition, de localité et d'induction. Le travail essentiel est de décrire un espace homogène de situations, gardé stable par le jeu élémentaire de règles admises de transformation, qui permet de rendre compte néanmoins de toute la diversité des situations rencontrées, notamment grâce à des inductions génératives.

112. *LTI*, p. 302-304.

Il est à noter que ce travail de systématisation, dans les sciences naturelles, est largement pragmatique. Dans le travail de construction de théories systématiques à partir d'un champ de problèmes, et réciproquement dans la mobilisation des théories pour la résolution de problèmes particuliers, la *modélisation* représente une sorte d'« embrayage » permettant d'articuler problèmes et théories de manière souple.

Les modèles ont été décrits de manière très générique par M. S. Morgan et Morrison (1999) comme des *instruments de l'investigation*[113]. De manière plus précise, Varenne définit un modèle comme

> un type de construit formel possédant unité et homogénéité. Cette unité et cette homogénéité sont choisies de telle manière qu'elles permettent à ce modèle de répondre à certaines questions ou de remplir certaines fonctions précises de compréhension, d'action, de communication, de gestion ou de décision en rapport avec un objet, un système, un phénomène ou une situation observable, concevable ou imaginable[114].

Dans les deux définitions, les « questions » ou l'« investigation » jouent un rôle central : un modèle n'a de sens que vis-à-vis d'un problème qu'il s'agit de résoudre, aussi général soit-il. Comme le dit également le scientifique Barry Jones :

> Un modèle est la manière de préparer un problème en vue de l'investigation[115].

Modéliser n'est donc rien d'autre que construire un espace de problème, puisqu'il s'agit de décrire l'expérience par un ensemble de significations régies par des règles opératoires internes, à partir desquelles des relations nouvelles (donc ultimement des inférences) peuvent être établies. On modélise tout autant dans le cadre de problèmes pratiques que théoriques, mais dans le second cas, celui de l'explication scientifique, le modèle est la véritable fin de l'investigation. Il est la mise en évidence d'un réseau spécialisé d'inférences qui peut convenir à la résolution d'une large classe de problèmes.

La modélisation participe donc de l'investigation théorique, qui est également une activité constructive, en ce qu'elle vise à établir et vérifier un système de significations. Cela explique que pour de nombreux scientifiques, il n'y a qu'une différence de degré entre modèle et théorie, qui porte sur leur généralité, précision et confirmation[116]. Une théorie permettrait d'être mobilisée pour une classe très

113. (M. S. Morgan et Morrison 1999, p. 10).
114. (Varenne 2013, p. 299).
115. *A model is the way you set up a problem for investigation*. Entretien rapporté par (Bailer-Jones 2002, p. 282).
116. (ibid., p. 291-293).

large de problèmes, tandis qu'un modèle aurait en général une portée assez étroite.

Le fond commun des théories et des modèles est qu'ils sont tous des systèmes de significations. Cependant il ne nous semble pas qu'ils se distinguent par un degré différent de généralité, car une théorie n'est pas explicitement orientée vers la résolution de problème, ou encore « adaptée » à une problématique locale, aussi étendue qu'elle soit. Comme l'exprime Varenne à la suite de sa définition :

> Construit autour d'une structuration formelle unitaire, le modèle se distingue de la théorie mathématisée par son caractère de localité, par son adaptation préalable aux questions initialement posées et par son impuissance à mener directement à des théorèmes (énoncés démontrés vrais et valant en toute généralité dans un cadre théorique fixé)[117].

Une théorie, nous l'avons déjà dit, se présente comme neutre de toute intentionnalité et indépendante d'une classe donnée de problèmes, aussi large soit-elle, et en cela elle se distingue des espaces de problème et des modèles. C'est cette neutralité qui lui permet de prétendre à « des théorèmes vrais » ainsi que le note Varenne, car elle semble s'intéresser simplement « à ce qui est », indépendamment de tout autre intérêt et de tout contexte. Nous ne pouvons dans le cadre de cette étude entrer dans le vaste débat des relations entre les notions de théorie et de modèle, dont M. S. Morgan et Morrison (1999, p. 18-20) donnent un aperçu : les modèles permettent dans certains cas de construire des théories, d'en explorer des implications, de les tester ou des interpréter. En retour, les théories, dans l'interprétation sémantique à laquelle souscrivent Van Fraassen et Kuhn, sont des collections de modèles. Quoiqu'il en soit de ces relations, instrumentales ou constitutives, il semble qu'il soit nécessaire, ou simplement efficace, d'opérer une telle division du travail intellectuel entre des systèmes de significations contextuels, instrumentaux, et somme toute contingents que sont les modèles, et des systèmes de référence que sont les théories, et dont le rôle est d'intégrer de manière cohérente et pérenne le savoir contenu dans les premiers.

§ 182. La dynamique scientifique

Il est dans ces conditions compréhensible qu'on ait pu caractériser l'activité théorique comme désintéressée et séparée du monde de la pratique. C'est que, tout entière occupée à la tâche critique d'expliciter nos théories implicites et naturelles

117. (Varenne 2013, p. 299).

afin d'en contrôler les inférences, elle se détache de l'action, comme la signification se détache de l'expérience. Mais elle n'est elle-même qu'une autre forme d'action :

> L'inférence contrôlée, c'est la science, et la science est, par conséquent, une industrie hautement spécialisée. C'est une pratique tellement spécialisée qu'elle ne semble pas du tout être une pratique. Cette haute spécialisation est en partie la raison de l'antithèse actuelle de la théorie et de la pratique[118].

Une caractéristique centrale de la démarche scientifique est donc qu'elle construit des réseaux de représentations qui n'ont plus de lien évident avec l'expérience courante. Les faits qu'elle convoque par ses observations contrôlées et ses protocoles expérimentaux n'ont plus grand' chose à avoir avec les faits et les actions que connaît le sens commun. Cela est normal, car le contrôle de l'inférence implique justement de se décentrer de l'environnement qualitatif, naturel et social de chacun, vers un monde objectif, neutre, appauvri, qui se départit du langage des émotions, des sensations, des conventions sociales, etc. afin de n'exhiber que les seules conditions de validité de l'inférence[119].

L'erreur du scientisme, selon Dewey, est de croire que les systèmes de significations de la science représentent le monde réel, et que la situation dans laquelle chaque personne se meut n'est qu'une fiction, une apparence, dont il faudrait se débarrasser. Mais il n'y a en fait nul privilège ici, simplement un autre mode d'accès au monde, une autre « forme de vie », dirait Wittgenstein. Le monde de la science n'a lui-même de valeur que pour autant que ses significations sont éprouvées et mises en pratique – l'expérimentation étant une forme de pratique.

Cela ne veut pas dire pour autant que l'effort scientifique est soumis aux fins utilitaires des pratiques courantes des personnes humaines – un reproche d'utilitarisme souvent fait à Dewey[120]. Au contraire, de même que tout problème résolu débouche *aussi* sur une nouvelle manière de percevoir l'environnement, et d'y reconnaître de nouveaux problèmes, l'activité scientifique s'émancipe très vite des questions initiales qui lui sont posées pour générer son propre corpus de questions :

> Ce mode de pratique qu'on appelle l'activité théorique émancipe l'expérience – et permet un progrès constant. Peu importe à quel point les compétences spécialisées s'améliorent, nous restons limités par la constance et la fixité de nos fins. Un progrès significatif, un progrès qui est plus que technique, dépend de la capacité à prévoir des résultats nouveaux et différents et à organiser les conditions de leur réalisation.

118. MW 8 : 78.
119. MW 8 : 81.
120. Voir à ce sujet les mises au point de Madelrieux (2016, p. 134-135).

> La science est l'instrument d'accroissement de notre technique pour atteindre des résultats déjà connus et appréciés. Plus important encore, c'est la méthode pour nous émanciper de notre esclavage à des fins coutumières, à des fins établies dans le passé[121].

En d'autres termes, l'objet de la science, qui est le contrôle de l'inférence, par exemple de A à B, ne permet pas seulement de rendre celle-ci plus sûre, mais également de découvrir des inférences de A à C, D, E,…termes qui sont inventés par là même et nous ouvrent de nouvelles possibilités d'action et d'expérience. La science ne répond pas seulement à nos questions et à nos problèmes, mais les reformule et en invente de nouveaux : elle construit des théories en ce qu'elle élargit et reconceptualise les théories naïves du sens commun.

Cette dynamique se voit le mieux dans le double rapport qu'entretient la science au sens commun. D'une part, elle est une reconstruction systématique des réseaux de significations du sens commun, et non pas une fondation nouvelle sur des terres étrangères. Dewey reprend fréquemment l'idée classique, héritée d'Aristote, que les sciences se sont développées à partir des techniques. Une théorie n'est donc jamais vraiment construite, mais toujours reconstruite à partir d'une structure antérieure. Cette opération est d'abord destructive (Dewey dit aussi : *désintégrative*[122]) des agrégats de signification antérieurs avant d'être reconstructive. L'exemple ci-dessus de la définition des métaux est éclairante : c'est parce que les opérations faites avec les métaux s'étaient élargies, qu'on avait par exemple commencé à remarquer qu'ils réagissaient différemment avec certains composés chimiques, qu'il a été nécessaire de revoir progressivement leurs caractérisations jusqu'à réorganiser la hiérarchie de leurs propriétés et à les relier entre elles en un nouveau système, qui correspondait mieux aux nouveaux problèmes de la science.

D'autre part, cette reconstruction n'est pas limitée à un simple réagencement des structures sur un terrain donné une fois pour toutes. Elle est également ouverture de perspectives, agrandissement, décentrement, si bien que la bonne métaphore architecturale, à cet égard, pourrait être celle des successifs agrandissements et embellissements du château de Versailles sous Louis XIV, recouvrant et annihilant le modeste pavillon de chasse qui fut à l'origine du projet. C'est la conception de cette double dynamique de reconstruction et d'agrandissement des systèmes de

121. MW 8 : 81.
122. *LTI*, p. 81.

significations qui permet à Dewey de suggérer une interprétation des révolutions scientifiques qui fait écho, par anticipation, à celle de Kuhn :

> L'histoire du progrès scientifique réel est marquée par l'adoption et l'invention de dispositifs matériels et de techniques apparentées [...] Les données nouvelles ainsi instituées font bien plus que fournir des faits pour confirmer et affiner les anciennes conceptions. Elles instituent *un nouvel ordre de problèmes* dont la solution nécessite *un nouveau cadre de référence conceptuel*. En particulier, c'est par l'utilisation de nouveaux instruments et techniques que des changements et des relations dans ces changements ont été révélés dans ce qui était auparavant considéré comme fixe ; un processus qui s'est poursuivi à un rythme accéléré depuis le XVII^e siècle [123]. *(Nous soulignons)*

10.5 Systèmes symboliques et systèmes objectifs

La section précédente a permis d'établir que les systèmes théoriques sont des systèmes de significations *contrôlés* pour assurer la validité systématique de l'inférence. Il s'agit à présent de montrer que, de manière similaire, les ensembles techniques sont des systèmes de ressources et de règles contrôlées pour guider l'action et assurer son succès dans un champ donné de situations envisagées. Les systèmes techniques sont donc tout autant perméables aux mathématiques que les systèmes scientifiques. HICKMAN (1992) a déjà montré que Dewey avait une conception technologique de la connaissance, mais ici on veut s'interroger plus spécifiquement sur le rapport qui en découle entre la forme pure de la connaissance que sont les mathématiques et les techniques concrètes, et notamment celles qui ont affaire aux machines. Nous revenons d'abord sur la notion de système objectif (§ 183) qui dénote d'abord les institutions et les vastes ensembles techniques, notamment les infrastructures. Cela nous permet d'étudier une longue analogie que Dewey propose entre mathématiques et machines (§ 184) et de proposer un premier rapprochement entre les concepts de *machine* et de *système* (§ 185).

§ 183. Systèmes symboliques et systèmes objectifs

On se rappelle que, dès sa réintroduction à la fin du XVI^e siècle, *système* a désigné aussi bien des ensembles organisés de connaissance (notamment chez les

123. *LTI*, p. 389.

encyclopédistes baroques et les théologiens protestants) que des ensembles objectifs au comportement réglé, en premier lieu le système des planètes mais bientôt également les systèmes organiques (par exemple le système digestif) ou institutionnels (les systèmes de gouvernements)[124]. Son emploi s'étend au XVIII^e siècle aux dispositifs techniques et aux infrastructures, comme les « systèmes de fortifications » ou le « système général de routes et de communications[125] ». Cette section vise à comprendre l'unité de sens qui sous-tend ces différents emplois du terme avec celui déjà dégagé pour les systèmes théoriques.

Avant d'en venir à Dewey, il faut remarquer que cette unité profonde avait déjà été pressentie au XVIII^e siècle par Johann Heinrich Lambert (1728 - 1777) dans sa réflexion sur la *systématologie*, une science qu'il visait à inventer dans les années 1760. Notre revue historiographique du chapitre 7 a passé sous silence ce mathématicien et philosophe, correspondant célèbre de Kant, car, s'il publia une *Architectonique des premiers éléments de la connaissance*, sa réflexion spécifique sur l'idée de *système*, le *Fragment zur eine Systematologie*, ne fut pas publiée de son vivant. Sa perspective quasi-formelle et visionnaire sur cette notion nous semble le placer en tous cas hors des cadres modernes. L'éditeur de ce texte considère que la réflexion de Lambert le positionne en précurseur direct de la théorie générale des sytèmes de Bertalanffy, et il n'est pas difficile d'y souscrire[126].

Lambert fait d'abord une distinction utile entre les systèmes selon la nature de leur « force de liaison ». Les systèmes intellectuels sont liés par la réflexion, les systèmes institutionnels sont liés par la volonté, et les systèmes physiques par les forces de la nature. L'intérêt de la réflexion de Lambert est de penser que, quelque soit la distance conceptuelle qui sépare ces systèmes, et qui pourrait faire croire à un simple rapport homonymique ou métaphorique entre eux, ils disposent des caractères communs de forme qui les rend passibles d'une étude commune. Par ailleurs, en tant que ces « forces » interagissent entre elles, il y a des systèmes mixtes, et notamment ceux des « moyens et des fins » auxquels il accorde une importance particulière (nous y revenons).

On trouve chez Lambert plusieurs distinctions capitales qui nous intéresseront par la suite[127], comme par exemple celles entre systèmes rigides (qui doivent être modifiés explicitement si leurs conditions changent, comme par exemple une mai-

124. Voir plus haut, § 126.
125. (VÉRIN 1993, ch. 7).
126. *Texte zur Systematologie*, éd. Siegwart, introduction, p. LXXIV.
127. Voir plus loin, chapitre 12.

son ou les machines), homothétiques (qui peuvent supporter accroissement et diminution de parties homogènes, comme par exemple une cité ou une compétence) et adaptatifs (pour qui le changement est naturel et qui convergent vers une position d'équilibre en fonction des conditions externes, comme un organisme ou une armée). Il distingue également entre systèmes transitoires (qui disparaissent avec l'atteinte de leur finalité) ou permanents (dont la finalité perdure dans le temps).

Concernant les systèmes téléologiques (de moyens et de fins), il en montre le caractère modulaire (des systèmes peuvent être composés de sous-systèmes), ce qui en fait une notion récursive, et distingue deux relations possibles entre sous-systèmes, l'une séquentielle (il y a un moyen immédiat auquel tous les autres sont subordonnés) et l'autre interactionnelle (tous les moyens concourent ensemble à produire la fin désirée). Lambert s'intéresse en particulier à la conception de tels systèmes. La conception est synthétique quand le problème est de déterminer à quelle fin pourrait servir un système : il faut pour cela en concevoir tous les effets possibles, et sélectionner ceux qui sont désirables. La conception est beaucoup difficile lorsque le problème est analytique : comment concevoir un système qui serve une intention donnée ? C'est ici que Lambert a l'intuition que les systèmes déjà construits (dont on a une connaissance synthétique) peuvent servir d'éléments et de modèles aux systèmes à construire. Les systèmes téléologiques peuvent donc naître et naissent en général dans une écologie de systèmes[128].

Cette unité de sens pressentie par Lambert peut être comprise dans la perspective de Dewey si on s'aperçoit que celui-ci ne fait que la retourner, et la garde par ailleurs intacte. Loin en effet que les systèmes de fins et de moyens soient dérivés de la coopération des systèmes de la connaissance, de la nature et de la volonté, ce sont ces trois là qui ne sont que des spécialisations de systèmes de significations – les significations n'étant rien d'autre, on s'en souvient, que des règles opératoires, des couplage moyen / fin unitaires. Les institutions et les ensembles techniques organisés sont ainsi, tout autant que les théories, des systèmes de significations eux-mêmes – idée qu'il nous faut à présent développer.

Dewey emploie régulièrement *système* pour évoquer les institutions comme la police, la justice, l'école. Il s'en justifie lors d'une longue analyse du chapitre 5 d'*Expérience et Nature*, à l'occasion d'une discussion sur l'objectivité des significations. Dewey essaie de montrer que sa doctrine établit une juste position médiane entre les réalistes, qui pensent que les significations constituent l'essence des

128. ibid., p. 137.

choses, et les nominalistes, qui pensent qu'elles sont des conventions arbitraires, instituées par les communautés humaines pour conduire leurs affaires. La thèse de Dewey est que les significations dénotent des opérations possibles et leurs conséquences, et qu'à ce titre, si elles sont en effet choisies pour leur intérêt, elles n'en sont pas pour autant subjectives, car elles évoquent des expériences réelles et objectives, inscrites dans une réalité organique et culturelle donnée. Cherchant des domaines illustrant de manière manifeste cette double nature des significations, Dewey recourt d'abord à l'analyse des codes de conduite et des lois, qui aboutissent à un « système légal de significations[129] », puis développe une analogie entre les mathématiques et les machines. Ce texte permet donc de bien saisir la liaison que voyait Dewey entre les systèmes de significations symboliques et les systèmes objectifs, institutionnels et techniques.

Dewey part d'un exemple frappant, celui du coup de sifflet d'un agent de la circulation, au son duquel véhicules et piétons règlent immédiatement leur comportement. Il s'agit d'un signe institué qui génère des attentes immédiates, du fait de l'habitude acquise de tous les acteurs, par exemple que les véhicules s'arrêtent et que les piétons passent. Il peut également être l'objet d'inférences, que celui qui ne s'arrêtera pas aura un accident, ou sera passible d'une amende ou de l'emprisonnement. En bref, à travers ce signe, c'est tout l'arrière-plan pratique de l'interaction sociale qui transparaît, et dont les conséquences sont bien objectives et éminemment réelles. La régulation sociale par ce type de signaux n'est pas nécessairement écrite, mais elle peut être l'objet d'une systématisation et d'une généralisation, et cela devient le droit. Le droit est donc bien un système de significations symboliques, mais il a une contrepartie, qui est un système d'inférences intégrées aux habitudes des agents, qui fait qu'ils savent comment se comporter même s'ils n'ont jamais lu le code civil.

Les institutions sont donc bien des systèmes en ce que nous disposons d'un ensemble d'anticipations et d'habitudes d'inférence à leur égard, bien précises, et en général fiables. On sait comment se comportera un policier dans telle ou telle circonstance (ou plutôt : on s'attend à ce qu'il se comporte de telle ou telle manière), ce qui se passera au commissariat, voire comment se déroulera une arrestation et quels sont les droits des prévenus. On peut avoir de tout cela une connaissance purement inférentielle, c'est-à-dire qu'on a des attentes sur ces sujets, qu'on sait plus ou moins bien les formuler, sans pour autant en connaître les fondements

129. LW 1:155.

juridiques détaillés et les définitions précises.

On pourrait objecter à cette distinction que les règles institutionnelles sont exprimées dans le langage, et qu'elles sont donc bien, à cet égard, symboliques. Il est difficile en effet d'imaginer qu'on ait pu développer un système d'inférences relativement cohérent et fiable à propos d'un ensemble phénoménal sans l'intermédiaire du langage. Dewey aborde explicitement cette question dans la *Logique* et, tout en notant son caractère spéculatif, il conclut en effet à la nécessité du langage pour que des *réseaux* de signes naturels puissent se mettre en place[130].

Cependant le point essentiel est qu'un système objectif ne dépend pas en général du langage pour son fonctionnement effectif. Il est dénommé tel d'abord par l'expérience que nous en avons, et par les attentes qu'il provoque en nous lorsque nous sommes confrontés à lui. Je n'ai pas besoin de me représenter l'article du code civil concernant le vol, ou même un impératif formulé plus simplement, pour savoir que je ne dois pas prendre des articles dans un magasin sans payer. Cette habitude a été inculquée en moi dans mon plus jeune âge, sans doute par de nombreuses indications symboliques, mais celles-ci ne sont plus nécessaires aujourd'hui pour diriger mon comportement. Le régime normal de l'action, en effet, est celui de l'habitude, et les systèmes d'inférences objectifs sont les « moules » de ces habitudes – à la fois façonnés par elles, mais aussi les façonnant activement, cette interaction étant le fait même de la transmission culturelle humaine. De la même manière que les inférences symboliques permettent de contrôler et de modifier les comportements habituels en donnant une signification précise à une situation problématique, il faut également concevoir les systèmes symboliques, non seulement comme les « moules » de ce type d'inférences, mais également comme des dispositifs qui peuvent agir sur les systèmes objectifs. Ainsi le droit écrit permet de réformer les pratiques coutumières.

Il y a donc un va-et-vient naturel entre systèmes objectifs et symboliques. Un système objectif peut être développé en un système de significations (par exemple un ensemble de règles coutumières qui deviennent des lois) ou, au contraire, être construit à l'aide de théories, comme lorsqu'on confie l'écriture de la constitution d'un État à des juristes étrangers.

On pourrait ainsi définir un système objectif comme *un ensemble de phénomènes, persistant dans le temps, dont l'évolution et les interactions internes et externes sont contraintes par des règles*. Ces règles, bien entendu, sont celles que

130. *LTI*, p. 61-62.

projette sur cet ensemble l'agent qui l'observe et interagit avec lui, à partir des inférences qu'il peut faire à son propos.

Cette définition, établie à partir d'une analyse des systèmes institutionnels, peut être étendue aux systèmes techniques. Une machine, comme un mécanisme, peut être qualifiés de *système* – ainsi un système d'air conditionné, un système de freinage – en ce qu'on insiste sur l'opération réglée et coordonnée de leurs parties. Les décrire, c'est expliquer immédiatement ce que fait chaque partie, comment elle influence telle autre, et ainsi de suite, jusqu'à ce que l'ensemble génère l'effet final attendu. On décrit donc une suite d'inférences qu'il est autorisé de faire, en faisant appel de manière sous-jacente à un système de significations donné, par exemple lorsqu'on explique le cycle de compression du liquide réfrigérant d'un système d'air conditionné à partir des principes de la thermodynamique.

On peut s'étonner de cette définition. Quel rapport a-t-elle en effet avec notre analyse de la systématicité ? En quoi un système objectif, ainsi défjni, permet-il de poser toute sorte de problèmes dans un champ donné, et de les résoudre de manière systématique ?

La réponse à cette question peut être rendue visible à partir d'exemples. De l'idée d'un réseau autoroutier il est possible de dériver tous les itinéraires possibles qui le parcourent. D'un organisme vivant, il est possible de déduire (au moins en théorie, et si l'organisme est suffisamment décrit) les valeurs limites de son métabolisme. L'idée architectonique d'un système objectif est celle qui permet d'organiser l'espace de ses comportements – qui ne sont rien d'autre, d'ailleurs, que celui des raisonnements que l'on peut faire à son sujet, sous les contraintes des règles qui le définissent.

Autrement dit, la représentation d'un système objectif fournit comme une minithéorie de son comportement, c'est-à-dire un espace adéquat dans lequel représenter une trajectoire possible qu'il pourrait exhiber dans le cadre de la résolution d'un problème. Une représentation du réseau autoroutier permet de résoudre la question de l'itinéraire le plus court de Bordeaux à Marseille, comme celle d'une bactérie la question de son rythme de reproduction dans tel biotope. Les systèmes objectifs sont des *systèmes de contraintes* portant sur le comportement des phénomènes cibles, ou encore comme des espaces de comportements possibles. De bonnes illustrations de systèmes objectifs sont de ce point de vue les jeux. Ils consistent en règles que doivent respecter tous les joueurs, mais qui les laissent par ailleurs libres de choisir leur comportement. Un jeu est intéressant lorsqu'il

parvient à établir un certain équilibre entre contraintes et initiatives laissées au joueurs, laissant éventuellement une certaine part au hasard.

Il faut ici remarquer qu'un système objectif n'est presque jamais indifférent à son environnement – sinon dans des cas-limites idéalisés. Il persiste pour autant qu'il a un comportement *intéressé* à sa survie. Il ne s'agit pas là de projeter des intentions sur un réseau autoroutier, mais il fait partie de son idée qu'il doit être construit solidement, entretenu, équilibré financièrement, etc. car il est construit pour la longue durée. Un système vivant peut également avoir d'autres intérêts que celui de sa propre subsistance, celle de sa famille, ou encore celle de son espèce, etc. Ainsi il est possible de qualifier progressivement l'espace de problème d'un système objectif en lui ajoutant des contraintes externes auxquelles il doit obéir.

Ces considérations nous seront utiles pour le dernier chapitre de notre travail, lorsqu'il s'agira d'étudier la programmation de systèmes informatiques interactifs.

§ 184. Machines, institutions et mathématiques

On parle le plus souvent de *système* pour désigner de vastes ensembles, comme le *système de production* d'une grande entreprise et le *système pénitentiaire* d'un État. Ces vastes ensembles s'apparentent aux systèmes de significations par une forme de générativité relative à leur domaine, c'est-à-dire par le fait qu'ils ne sont pas conçus spécifiquement pour répondre à telle ou telle situation, mais qu'ils doivent s'accommoder de toutes. Le système de production d'une entreprise désigne l'ensemble de ses usines, de ses sous-traitants, de ses fournisseurs, de ses réseaux logistiques et de ses savoir-faire, en tant qu'ils sont capables de répondre de manière générique à des besoins de production variés, incluant notamment un flux continu d'innovations auxquelles s'adapter. Ici le système de règles qui les régit apparaît bien plutôt comme un système de moyens plutôt que de contraintes, même si les deux termes sont conjugués l'un à l'autre. Ainsi, sur le plan institutionnel, les systèmes objectifs dénotent souvent des cadres de gouvernance et d'échange entre les personnes (marchés, organisations, institutions étatiques, réseaux de relations sociales et culturelles, etc.) et, sur le plan technique, des infrastructures organisées en réseau – on parle ainsi du système ou du réseau autoroutier, du système téléphonique, du système fluvial, etc. – qui mettent à disposition des usagers des moyens génératifs pour résoudre leurs problèmes particuliers (les segments du réseau sont connectés les uns aux autres et permettent ainsi de composer des segments plus complexes), tout comme les systèmes de signification symboliques, et en parti-

culier les théories scientifiques, mettent leurs concepts et leurs règles génératives au service de tout type d'investigations pour résoudre des problèmes particuliers. Du fait de cette générativité, les infrastructures héritent également des systèmes de significations leur visée systématique. Cette visée s'exprime dans le droit des infrastructures par la notion de service universel, et certaines infrastructures sont d'ailleurs qualifiées de service public et nationalisées. Outre les infrastructures, on doit aussi inclure parmi les systèmes objectifs les chaînes techniques opérationnelles conçues pour fonctionner ensemble (ainsi par exemple la chaîne de valeur (technique) de l'industrie automobile, qui définit des standards et des normes permettant d'assurer l'interopérabilité des composants).

Les systèmes objectifs semblent donc avoir une expansion plus malléable que les systèmes de signification quant à la portée des problèmes qu'ils permettent de résoudre. Dans le cadre d'un système autoroutier, on peut exposer une très grande quantité de problèmes, comme par exemple, aller de Paris à Nice, ou de Bordeaux à Lyon, et trouver en général plusieurs manières de les résoudre, et il s'agit donc bien d'un espace génératif[131].

On pourrait objecter que le système de freinage d'une voiture résout un seul problème : freiner efficacement le couple moteur. Le comportement de ce « système » est d'ailleurs très restreint, et en général réduit à quelques degrés de liberté. La théorie du système de freinage, si on essayait de l'expliciter de manière isolée, serait très pauvre. Il ressemble bien plutôt à une procédure qui résout un problème particulier qu'à un système générique.

Cependant, ce qui est systématique dans un mécanisme comme le système de freinage n'est pas l'objet lui-même, mais le vaste arrière-plan théorique et technique qui l'a conçu ainsi que ses pièces, qui a analysé leur performance individuelle et commune, permis leur fabrication, etc. Le système de freinage, comme le système électrique, ou le système d'air conditionné, ont comme point commun de ne pas désigner des objets du point de vue de l'utilisateur, qui parlerait plutôt directement de freins, d'électricité, et d'air conditionné, mais du point de vue du technicien ; ils sont cachés du premier, qui ne doit pas s'en préoccuper : *système* dans ces expressions connote ici la complexité et l'inaccessabilité du mécanisme aux personnes non-initiées.

Cette interprétation, que tout système technique est dénoté tel par référence implicite au système technicien – pour détourner l'expression de Jacques Ellul –

131. Sur le sens que nous donnons ce terme, voir plus haut § 149.

dont il est issu, nous permet d'accéder à une vaste analogie entre mathématiques et machines par laquelle Dewey prolonge son analyse du développement du système judiciaire. Cette dynamique interne des règles et des procédures légales qui sans cesse s'accroissent pour tenir compte de nouvelles situations et, ce faisant, se reformulent et se réforment parfois en profondeur – cette dynamique interne n'a rien de spécifique au système judiciaire. Elle permet, dit Dewey, de comprendre par analogie celle du développement incessant des mathématiques :

> Il n'y a donc rien d'étonnant à ce que les significations exprimées par des symboles soient capables de produire un système vaste et croissant de mathématiques. Une essence, qui est une méthode procédurale, peut être liée à d'autres méthodes procédurales pour donner lieu à de nouvelles méthodes ; pour amener une révision des anciennes méthodes et former un tout systématique et ordonné – sans qu'il y ait aucune référence à quelconque application à un ensemble particulier d'existences concrètes, et dans l'abstraction complète de toute conséquence particulière[132].

Dewey passe alors immédiatement à la seconde analogie, qui nous intéresse particulièrement. Elle porte sur le monde de la technique, et en particulier sur les machines, dont il identifie l'aspect essentiellement relationnel, qui explique le progrès technique :

> La comparaison avec des machines comme une moissonneuse-lieuse ou un système téléphonique est utile. Les machines ont évolué au sein de l'expérience humaine, pas avant elle ou indépendamment d'elle. Mais elles sont objectives et contraignantes au regard de processus physiques et psychiques particuliers actuels ; *ce sont des méthodes générales pour atteindre des conséquences* ; ce sont des interactions d'existences physiques préexistantes. […]
>
> Lorsque les machines ont atteint un certain stade de développement, les ingénieurs peuvent se consacrer à la construction de nouvelles machines et à l'amélioration des anciennes sans référence spécifique aux utilisations et aux applications concrètes. C'est-à-dire que les inventeurs sont guidés par la logique inhérente aux machines existantes, par l'observation de la cohérence des relations que les parties de la machine entretiennent entre elles et avec le modèle de la machine entière. Une invention peut ainsi résulter de calculs purement mathématiques. Néanmoins la machine reste une machine, un dispositif instrumental de régulation des interactions en référence aux conséquences.[…]
>
> Lorsque le « concept » d'une machine, sa signification ou son essence incarnée dans un symbole, génère de manière déductive des plans de nouvelles machines, l'essence est féconde car elle avait été, dès le début, conçue dans un but. […] Si nous

132. LW 1:156.

> suivons l'exemple des cas empiriquement vérifiables, il apparaîtrait alors que les essences mathématiques et morales peuvent être dialectiquement fécondes, parce que, *comme d'autres machines*, elles ont été construites dans le but d'assurer certaines conséquences avec le minimum de gaspillage et le maximum d'économie et d'efficacité[133]. *(Nous soulignons)*

Dans ce texte très riche, les machines individuelles sont vues comme des parties d'un système objectif plus vaste, le système technicien qui les sous-tend et qui est animé de la même dynamique de progrès que les systèmes institutionnels et les systèmes de significations. Les inventions dans ces trois domaines s'expliquent par un même fait fondamental, que toute opération au sein d'un système donné entre « en dialogue[134] » avec les autres opérations de ce système, qui met en évidence de nouveaux problèmes ou de nouvelles opportunités. De nouvelles machines naissent d'anciennes machines, comme de nouveaux concepts mathématiques naissent d'anciens. Il serait vain d'en rechercher une origine première, car mathématiques et technique ne sont pas des domaines séparés de l'expérience humaine, mais « ont évolué au sein d'elle », elles ne sont rien d'autre que des visées systématiques de l'expérience, de l'accomplissement des fins.

Mais il y a plus : Dewey rapproche explicitement les machines des concepts mathématiques et des règlements judiciaires en unifiant toutes ses notions sous celle de « méthode », qui doit être identifiée à l'« essence » dont parle la métaphysique classique. Les mathématiques inventent et décrivent les méthodes procédurales dans leur pureté – sans référence à leurs applications possibles dans l'expérience, alors que la technique invente des méthodes « attachées » à des instruments qui régulent des interactions concrètes. Est donc pointée ici une co-appartenance originaire des mathématiques, des techniques et des institutions. La machine peut être conçue mathématiquement et elle se prête par excellence au traitement symbolique (troisième paragraphe). Mais réciproquement, les essences (c'est-à-dire les procédures) mathématiques et judiciaires sont qualifiées de « machines » à la fin du texte.

§ 185. Machines et systèmes

Nous pouvons aller plus loin. Il n'est sans doute pas fortuit que Dewey recoure à l'exemple d'un système de *machines* lorsqu'il veut illustrer le progrès technique.

133. LW 1:157.
134. LW 1, 152.

Certes, il est devenu naturel, depuis la révolution industrielle, d'associer progrès technique et invention de machines, cependant notre analyse permet d'en donner une interprétation plus approfondie.

Nous avons déjà suggéré l'idée qu'une *géométrie des machines*, si on en développait pleinement le concept, aurait sans doute des propriétés systématiques éminentes. Cette remarque ne se fonde pas seulement sur l'observation que les machines jouèrent un rôle majeur dans l'émergence des systèmes scientifiques modernes, comme nous l'avons rappelé[135]. Elle se fonde également sur quelques observations faites à la partie précédente dans le domaine de la technique[136]. En effet, le concept de *machine* – dont l'emploi *industriel* devient une réalité observable au XVIII^e siècle – va être décisif pour que la possibilité d'une systématicité de la délibération technique devienne envisageable. D'abord, la machine permet de rendre exacte et absolue la division du travail entre conception et exécution, l'une étant le lieu de l'invention créatrice de la solution, l'autre celui de sa mise en application exacte. Cependant, une machine, prise isolément, n'a rien de systématique. Elle est au contraire « rigide » et par là, fragile : elle ne sait exécuter que le programme pour lequel elle a été conçue, et ne dit pas d'elle-même quelles sont les situations où elle réalisera sa fonction, et celles qu'elle exclut de son champ d'application. C'est par son intégration à des écosystèmes techniques qu'elle peut devenir systématique.

Or c'est là qu'une seconde propriété remarquable de la machine entre en jeu : étant l'objet de la géométrie du mouvement, elle est un *opérateur de récursivité*, parce que le mouvement d'une machine peut servir à en construire une autre[137]. Ou encore, en visant de confier certaines actions (résolvant un problème donné, comme le transport de marchandises) à une machine, on créée un problème de second degré, celui de la construire, et ainsi de suite. La machine *peut* créer son propre écosystème technique, et même *doit* le faire, comme Marx l'avait très bien vu, en en tirant toutes les conséquences :

> La grande industrie fut donc obligée de s'emparer de son moyen de production caractéristique, la machine, et avec des machines, de produire des machines. C'est seulement à partir de ce moment qu'elle se créa sa base technique adéquate, en se dressant sur ses propres pieds et en devenant elle-même[138].

La capacité des machines à produire d'autres machines permet au système indus-

135. Voir plus haut, sections 7.4 et 7.5.
136. Voir plus haut, section 7.3.
137. Voir plus haut, fin du développement § 119.
138. (Marx [1867] 2006, p. 431).

triel de devenir *autonome*, de « devenir lui-même » dit Marx. Il nous semble y avoir là un schéma logique général, qui vaut au-delà du domaine industriel. Les notions de *machine* et *système* (ce dernier terme pris dans son sens objectif[139]) sont fortement couplées, et l'une apparaît comme la duale de l'autre.

Certes, à première vue, *système* semble être un terme bien plus générique que *machine*, s'appliquant à toutes sortes d'objets qui n'en sont pas – comme des systèmes de routes, d'institutions politiques, d'organes, de planètes, etc. Par ailleurs, *système* semble aussi plus malléable, puisqu'il peut désigner une esquisse de machine (comme on l'a vu à propos des systèmes qu'on pose dans l'analyse des problèmes scientifiques[140]), l'infrastructure requise à la production de machines (comme un système technique), mais aussi un assemblage de machines, ou une théorie de machines, etc.

C'est cependant quand les systèmes portent sur des machines qu'ils arborent le plus nettement leur propriété fondamentale d'être *aussi* des structures récursives, c'est-à-dire d'être composés d'autres sous-systèmes ou d'être engendrés par des infrastructures systémiques, pour ne citer que ces deux relations de récursivité qui nous intéressent ici. La machine, en tant qu'elle est une structure déterminant un processus d'usage, et engendrée par un processus de construction, semble bien être le pivot élémentaire de telles relations, situé aux *articulations* de ces systèmes et qui les rend donc possibles.

Si on tente de transposer cela au domaine de la logique, où le système est d'abord un ordre de propositions ou de significations liées par des inférences, chaque liaison peut être vue comme une micro-machine *effectuant* la démonstration requise pour passer d'une signification à l'autre[141]. Dire qu'un système est autonome, comme Marx le dit de la grande industrie, ne signifie pas seulement qu'on peut démontrer certaines propositions à partir d'autres propositions déjà démontrées, comme on peut assembler des machines à l'aide de machines plus élémentaires. Cela signifie que des machines elles-mêmes peuvent procéder à cet assemblage.

La relation que nous venons de décrire entre *machine* et *système* permet de mieux mettre en perspective le schéma *construction / usage* qui caractérise les théories selon Ryle, et les systèmes en général. Nous l'avions rapproché, à l'oc-

139. Voir plus haut, § 126.
140. Voir plus haut, § 125.
141. Voir plus haut, § 119.

casion d'une citation de Bacon[142], des images économiques de l'investissement et du profit, ou encore de l'accumulation et de la rente, qui sont somme toute assez répandues dans les discours pédagogiques sur la connaissance et la recherche. On peut remarquer à présent qu'il s'agit plus que d'une simple métaphore. Cette structure caractérise également la machine qui, elle, a un rapport bien réel au capital. Il n'y a nul mystère dans ce parallélisme. *Système* et *machine* sont les objets respectifs des deux formes d'investigations qui se cristallisent enfin à l'époque moderne : la construction de systèmes de connaissance et la résolution systématique de problèmes. Ces objets sont *conçus pour* durer, s'accumuler, et servir à nouveau, même si la machine / solution a une portée seulement locale et souvent temporaire relativement au système, qui en est l'infrastructure. Ensemble ils sont les concepts élémentaires de l'économie de la connaissance.

10.6 Synthèse

Nous concluons cette partie en tentant de formuler, de manière synthétique, les propositions principales qui ressortent de notre enquête épistémologique sur la résolution de problème. Avant de ce faire, il est utile de reprendre les différentes formes qu'ont prise, au cours de ce parcours, *nos deux questions directrices* concernant l'effectivité de la connaissance, la première à propos de la nature de la résolution de problème et de sa situation entre connaissance théorique et pratique, et la seconde à propos de la possibilité pour la connaissance elle-même d'être objet de procédures et de traitements mécaniques.

La première question, telle que nous l'avons formulée en introduction[143], porte sur le rapport qu'entretient la connaissance de *ce qu'il faut faire* pour résoudre tel ou tel problème avec la connaissance simple de *ce qui est*. Nous nous sommes même étonné, un peu plus loin[144], qu'on appelât *connaissance* cette compétence très spéciale qu'ont les personnes humaines de *concevoir* des solutions de situations problématiques – cela ne nous semblait avoir aucun rapport avec la connaissance au sens classique.

Nous avons entamé à proprement parler cette recherche au début de la

142. Voir plus haut, p. 548.
143. Voir plus haut, § 14.
144. Voir plus haut, § 19.

deuxième partie[145], après notre analyse contractuelle de la programmation, qui la présentait (pour le dire rapidement) comme une résolution de problème à l'aide d'une machine. Nous nous sommes alors demandé tout simplement ce qu'était la *résolution de problème*. Cette question est devenue plus complexe encore lorsqu'il est apparu, d'une part qu'il était difficile de distinguer les résolutions de problèmes « pratiques » (concrets, utilitaires) de celles de problèmes « théoriques » (intellectuels, scientifiques), mais d'autre part qu'il semblait y avoir une différence très réelle entre une *manière* pratique ou technique de les résoudre, et une manière systématique, habituellement associée à la posture scientifique, qui semblait également ordonner les méthodes industrielles et informatiques. Notre question est alors devenue, au début de la troisième partie[146], celle de savoir en quel sens il était légitime de distinguer entre savoirs théoriques et pratiques, et *où* situer la résolution de problème sur cette cartographie, ainsi que ses propres subdivisions.

La lecture de Kant, qui définit l'entendement comme le pouvoir des règles, nous a permis de rendre compte de l'indifférence de l'investigation à la *matière* théorique ou pratique du problème à résoudre ; cependant, au début de notre quatrième partie[147], un tel résultat était encore conditionnel à l'hypothèse que la dualité entre action et situation devait être comprise comme *interne* à la logique de la résolution de problème, et donc relative à chaque problème, et non comme relevant de catégories générales de l'intelligibilité des choses.

La deuxième question formulée en introduction[148] porte sur la capacité des procédures à augmenter notre capacité de connaître, ce qui semble signifier que la connaissance a le caractère d'un *faire*, d'une pratique susceptible de prescriptions et de techniques qui peuvent en augmenter l'effectivité.

Lorsque nous avons entamé notre recherche en début de deuxième partie, cette question est devenue plus étroite : nous nous sommes demandé comment il était possible qu'on résolve des problèmes à l'aide de machines puis, en troisième partie, à l'aide de systèmes. En début de quatrième partie, cette question s'est faite plus précise encore : nous nous sommes demandé comment il était possible, en mathématiques, de concevoir des systèmes organisés de règles qui, tout en étant finis, voire minimaux, permettent d'exprimer la richesse indéfinie des problèmes

145. Voir plus haut, § 62.
146. Voir plus haut, § 94.
147. Voir plus haut, § 144.
148. Nous omettons ici les renvois qui sont identiques à ceux des trois paragraphes précédents.

de la connaissance et de l'action, et de donner les outils pour les résoudre.

Nous sommes à présent en mesure de répondre de manière unifiée à ces questions. Cette synthèse a son creuset dans la théorie de l'investigation de Dewey que nous venons d'exposer, mais elle se nourrit également de nos lectures de Kant et de Martin-Löf. Elle a pour arrière-plan la tradition aristotélicienne de l'investigation ainsi que les réflexions des Temps modernes sur les notions de *machine* et de *système*. Nous établissons ainsi des rapprochements et des thèses qui complètent la théorie de Dewey dans des directions qu'il ne fait que suggérer.

Il n'y a pas ici de prétention à *fonder* une philosophie de la connaissance, car nous laissons indéterminée la nature des *actes, significations, règles*, etc. qui forment le treillis conceptuel de nos thèses concernant l'investigation. Ces termes prétendent seulement mieux se prêter à la description, *à gros grains*, d'expériences de réflexion que chacun peut faire. Il s'agit de montrer, d'une part, quelle est la logique unique et commune d'investigations qui peuvent de prime abord sembler essentiellement différentes, et d'autre part mieux en pointer les différences propres. Ce cadre descriptif reste compatible, selon nous, avec la plupart des options épistémologiques fondamentales[149].

Cette synthèse commence par rappeler en quel sens nous affirmons que les règles sont le contenu des significations, et que toute règle a immédiatement un contenu expérientiel et opérationnel (§ 186) ; cela permet d'éclairer le caractère transversal de la résolution de problème (§ 187). Nous rappelons ensuite la distinction que nous avons faite entre l'investigation elle-même et la procédure qui peut la récapituler (§ 188). Cela nous permet de décrire la manière habituelle, « pratique » de résoudre des problèmes, qui s'extériorise naturellement en techniques et en institutions (§ 189). La construction d'ensembles techniques et institutionnels est déjà partie de ce que nous avons appelé *investigation globale*, cependant celle-ci ne peut devenir proprement systématique que lorsqu'elle a pour objet la réforme d'ensembles de significations ; la forme systématique est essentiellement mathématique (§ 190). Ces cadres systématiques permettent à la résolution de problème de simplifier sa vérification, mais surtout lui apportent une garantie d'exhaustivité, qui permet à l'investigation de se supprimer elle-même, en devenant pure ratiocination (§ 191).

Nous avons conscience du double écueil que présente cette synthèse : d'une part, comporter d'inévitables redites, notamment au regard de la doctrine de De-

149. Voir plus haut, § 3.

wey que nous venons d'exposer, et d'autre part, comporter des formules concises pouvant donner l'impression de raccourcis cavaliers. Cependant, il nous a semblé que de tels risques étaient compensés par l'utilité pour la lectrice de trouver rassemblées en un seul lieu et articulées entre elles des thèses positives établies progressivement au cours des trois dernières parties, qui vont être mobilisées dans la partie suivante afin d'achever notre élucidation des notions de *programmation* et de *machine*.

§ 186. Règles et significations

Le geste fondamental de Dewey est d'inscrire l'investigation, c'est-à-dire la résolution de problème, dans l'activité élémentaire de l'individu qui est de s'adapter à son environnement. Comme Aristote, Dewey voit dans la résolution de situations problématiques une activité commune à l'homme et à une large partie du règne animal. Cette résolution peut procéder, chez les animaux développés, par la considération des attentes qu'évoquent certains éléments dans la situation (par exemple de la fumée qui évoque la présence du feu). Il n'y a cependant *problème*, à proprement parler, et *résolution de problème*, que chez les individus dotés de langage, c'est-à-dire capables d'instituer des marques (que nous appellerons *symboles*) pour évoquer des expériences potentielles ainsi que les inférences permettant de passer des unes aux autres. Résoudre un problème est l'activité intellectuelle de représenter une situation problématique par des symboles et de manipuler ceux-ci, *ainsi que* leurs évocations, jusqu'à parvenir à l'évocation d'une situation satisfaisante, qui peut être appliquée[150].

Suivant Dewey, nous appelons *signification* cette évocation, par un signe institué, d'une expérience potentielle[151]. Une signification et les inférences qu'elle véhicule n'ont donc aucune effectivité par elles-mêmes. Elles ne disent jamais ce qui est, mais seulement ce qui pourrait advenir dans l'expérience, si telle action ou telle opération était réalisée – que cette opération soit un simple acte de l'attention, comme observer telle marque phénoménale, ou des actes plus complexes, comme mesurer une température, ou chauffer une éprouvette[152].

Il n'est pas cependant nécessaire de poser qu'il y a des significations et des expériences qu'on pourrait qualifier d'*élémentaires* – comme par exemple celles

150. Voir plus haut, § 167.
151. Sur le terme d'*évocation* ou de *suggestion*, voir plus haut § 170.
152. Voir plus haut, § 170.

qui seraient liées aux actes de perception. Ces derniers ne surviennent jamais que dans le contexte de situations problématiques où il est vital de *remarquer* certains de leurs caractères ; il faut par exemple un effort spécifique pour remarquer qu'un fauteuil est vert ou rouge, car en général une telle qualité est sans importance. Les significations ne sont donc pas nécessairement construites les unes sur les autres de manières hiérarchique, mais elles font plutôt *réseau*, elles se rapportent les unes aux autres comme des *constellations* de sens, selon l'expression de Dewey[153].

Une signification est d'abord une *règle*, ou un ensemble de règles, – ces règles mêmes dont Kant disait que le concept était leur représentation[154]. C'est dans l'idée même de *règle* que s'inscrit la dualité entre la situation (qui est l'objet des conditions et des conséquences de la règle) et l'action (qui est la représentation de l'effectuation de leur liaison) ; cette dualité a donc un sens logique. Son inscription possible dans l'ordre même des choses relève de l'ontologie et n'est pas nécessaire à notre propos[155].

Une règle porte, de manière ultime, sur une expérience qu'il est possible de *vivre*, si telles ou telles conditions sont réunies dans la situation, et si l'agent lui-même s'y trouve dans ou telle ou telle disposition. Une règle a donc toujours pour condition essentielle une forme d'action ou d'opération de l'agent, même si celle si est aussi simple qu'un acte de perception. Quand bien même la règle semble dire simplement que le coucou de l'horloge sortira à midi, il est fait référence ici à la possibilité, pour un observateur patient, d'observer ce coucou *s'il attend devant l'horloge* jusqu'à cette heure – ce qui suppose encore qu'il sait *attendre* et donc compter le temps, de la même manière que les Égyptiens avaient trouvé les règles des éclipses parce qu'ils avaient appris *à compter* les révolutions des astres[156].

Une signification, en tant que règle, *contrôle* la validité d'une inférence de telle expérience à telle autre. Pour prendre un autre exemple, la règle de ductilité des métaux me permet d'inférer, d'une tige métallique que je tiens à la main, à ma capacité à la tordre aisément si je l'approche du feu.

En tant qu'un prédicat comme la ductilité est conçu comme *interne* à un sujet comme le métal, leur relation est catégorique, pour reprendre le terme employé par Kant ; s'il était conçu comme participant de la situation (ici le feu), leur relation serait hypothétique, c'est-à-dire dépendante d'une condition *externe*. En effet, si

153. Voir plus haut, § 167 et § 181.
154. Voir plus haut, § 132.
155. Voir plus haut, § 138.
156. Voir plus haut, § 169.

le feu avait pour propriété de rendre malléable toute chose (et non parfois de les consumer, ou de seulement les chauffer) la ductilité ne serait sans doute pas perçue comme une propriété caractéristique des métaux, mais du feu lui-même. C'est pour cette raison que l'établissement de significations *disjonctives*, qui sont de second ordre et qui tentent de décrire *exhaustivement* les relations entre les significations disponibles concernant un certain champ de phénomènes, sont cruciales afin que des définitions et des lois stables puissent être établies à propos de ce champ. C'est par ces tentatives d'établir des significations disjonctives que commence l'effort systématique[157].

§ 187. Unité de la résolution de problème théorique et pratique

Une proposition n'est rien d'autre que l'explicitation d'une règle. Elle doit être affirmée comme valide dans l'expérience afin de devenir un jugement. Le jugement est un acte par lequel l'entendement s'*engage* à une inférence donnée au terme d'une investigation. Il *est* la résolution du problème, si on veut bien entendre dans le terme *résolution* aussi bien l'évocation d'une volonté déterminée que celle d'un conflit dénoué. Il est une décision qui porte sur la situation présente, qu'il ne fait pas que décrire, mais qu'il *modifie*, même si, par exemple, il consiste dans la simple prédiction d'un événement futur ou l'explication d'un fait passé. Attribuant à la situation (le sujet du jugement, le *X* de Kant) la proposition, qui entraîne avec elle tout un réseau de significations (son prédicat), il enrichit cette expérience de ce simple fait – en ce qu'il y montre de nouvelles connexions et de nouvelles actions possibles ; il crée une nouvelle situation qui prépare l'agent à de nouvelles intentions. Un jugement, même s'il n'est communiqué à personne, modifie la situation comme le ferait n'importe quelle action physique[158].

C'est de ce point de vue qu'il est possible de comprendre pourquoi la résolution de problème, aussi grande que soit la richesse de ses formes possibles, n'est pas divisée de manière essentielle entre problèmes *pratiques* ou *concrets* d'une part, et problèmes *théoriques* ou *intellectuels* de l'autre. La raison en est que ces deux domaines élémentaires du savoir – ce qui est et ce qu'il faut faire – sont d'emblée conjoints en chaque jugement et même en chaque signification[159]. Ainsi, décider du prix auquel négocier un achat de denrées agricoles dépend seulement d'une longue

157. Voir plus haut, § 133.
158. Voir plus haut, § 174.
159. Voir plus haut, § 174.

série de jugements concernant la météorologie, l'économie et la botanique : c'est que ces analyses portent sur la situation présente, qui est déjà grosse de l'avenir, et de laquelle le prix est aussi bien la représentation exacte que la décision à laquelle il faut se résoudre[160]. Réciproquement, juger de la vitesse d'une galaxie dépend avant tout d'un protocole impliquant plusieurs actions physiques et intellectuelles – récupérer les bonnes données d'observation spectrométrique, utiliser les bons modèles computationnels, vérifier les calculs grâce à telle ou telle méthode. C'est que toutes ces actions sont les moyens par laquelle une inférence complexe se construit progressivement. À la source de cette circulation libre des jugements sur ce qui est et des décisions sur ce qu'il faut faire dans la résolution de problème se trouve la co-appartenance intime de l'acte du jugement et de son objet, ou encore de la décision à prendre et de la situation dans laquelle on se trouve.

On objectera : il y a tout de même une différence de finalité entre la recherche de la vitesse d'une galaxie et celle du prix auquel acheter les olives, car la première ne modifie en rien la situation. Or rien n'est moins vrai, car l'entreprise de ce calcul est toujours inscrite dans un projet, quel qu'il soit, qu'il soit pédagogique, technique ou théorique. C'est au niveau de ce projet d'ensemble que peut seulement être jugée cette intention, et nullement au niveau du problème lui-même, dont la solution peut se prêter à toutes sortes de fins[161].

D'où vient alors la distinction si commune entre règles du jugement et de l'action ? Notre analyse de la géométrie euclidienne[162] suggère qu'il existe une division du travail assez naturelle – mais non obligatoire – entre actes d'observation et actes de construction, entre savoir analyser une situation et savoir quoi y faire. Cependant l'analyse d'une situation n'est nullement passive, elle mobilise au contraire des activités pures, celles par exemple de diriger l'attention vers les marques pertinentes de la situation, ou celles d'opérer des inférences afin d'y déceler des marques non immédiatement visibles (par exemple l'égalité d'angles éloignés). Réciproquement, savoir agir requiert que des conditions soient préalablement vérifiées. C'est par un double effort de la réflexion, celui d'apprendre à lire une situation et celui d'apprendre comment s'y comporter, que la personne est capable de développer des règles du jugement de plus en plus riches, dépendant, quant à leurs conditions, de capacités à discerner de plus en plus fines, et quant à leurs conséquences, de

160. Voir plus haut, § 136.
161. Voir plus haut, § 140.
162. Voir plus haut, section § 158.

moyens d'actions de plus en plus intégrés.

§ 188. Investigation et procédure

La résolution de problème procède de manière itérative. En décrivant la situation problématique par des significations, des actions et des pistes de solutions sont suggérées, qui sont vérifiées par le raisonnement ; ces analyses, lorsqu'elles sont négatives, suggèrent néanmoins de nouvelles pistes, la considération de sous-problèmes, voire la reconsidération du problème tout entier, et ainsi de suite. Petit à petit, la situation problématique est reformulée, jusqu'à ce que l'investigation parvienne (ou non) à une solution satisfaisante. Dans le cas positif, c'est l'ensemble des liaisons qui ont été établies entre la représentation de la situation problématique et celle de la situation satisfaisante qui constituent la solution – soit qu'ensemble elles convainquent l'investigateur de la validité d'un jugement (preuve) soit qu'elles l'aident à prendre une décision favorable s'il entreprend un certain cours d'action[163].

La plupart des investigations s'arrêtent là : la solution présentée par le raisonnement est acceptée par un jugement, qui est immédiatement une décision ayant des conséquences dans la situation (par exemple, l'investigateur passe à l'action).

Cependant, lorsqu'on s'attend à ce qu'un problème résolu survienne à nouveau (ou lorsque d'emblée l'investigation s'est attachée à la résolution d'une classe *générale* de problèmes, *une fois pour toutes*) il peut être utile d'exprimer cette solution par une séquence de significations s'enchaînant rigoureusement, ou plus généralement encore par un petit ensemble de règles permettant de diriger l'agent dans la résolution future de problèmes similaires. Cette procédure s'exprime, soit par les termes mêmes avec lesquels l'investigateur a mené son enquête (par exemple des schémas, des diagrammes), soit dans des termes différents, par exemple s'il veut la communiquer à autrui – que ces significations soient empruntées au langage courant, aux ensembles techniques dans lesquels il évolue, ou aux théories qu'il maîtrise.

Il s'agit là, en quelque sorte, d'une *seconde* investigation qui vient se superposer à la première ; nous appelons *procédure* son résultat, et *mise en procédure* ce type d'investigation. On peut se représenter ici l'élève qui, après avoir résolu son exercice de mathématiques au brouillon, recopie la solution retenue au propre,

163. Voir plus haut, § 173.

avec sa démonstration, en évitant soigneusement toutes les pistes inutiles initialement explorées. Nous nommons ce résultat par le terme générique de *procédure*, et qui peut porter aussi bien sur une preuve (ce qui emporte l'assentiment, *i.e.* qui transforme une situation intellectuelle) que sur une action (ce qui transforme une situation matérielle)[164].

Une procédure ne décrit pas l'investigation telle qu'elle s'est déroulée, ni même telle qu'elle se déroulera à nouveau à l'avenir, mais elle permet de la baliser afin qu'elle devienne plus efficace, qu'elle puisse être transmise et apprise, etc. C'est en ce sens que la procédure est le *fond* de la connaissance dans un jugement ou dans une décision. On peut toujours énoncer des propositions, ou émettre une préférence. Mais *juger* ou *décider*, à proprement parler, c'est-à-dire rationnellement, c'est agir en exhibant la justification de son acte qui permet à chacun (y compris soi-même), de contrôler sa justesse[165]. C'est en ce sens que, pour Dewey comme pour Kant et pour Martin-Löf, le raisonnement doit être pensé comme un jugement complexe[166] qui est déplié dans ses raisons internes pour prendre sa force et parvenir à l'assertabilité garantie. Cette interprétation s'oppose à celle, plus traditionnelle, qui assimile les jugements à des propositions acquérant leur force par leur position terminale dans un raisonnement qui les manipule et les agence entre eux comme une simple manière inerte[167].

Cette réflexivité possible de l'investigation sur elle-même peut encore se dérouler à un niveau supérieur, lorsqu'elle cherche non seulement à se récapituler, mais à saisir les règles de son mouvement lui-même, afin d'atteindre par là une généralité plus grande. Par là sont précisément explicitées les trois classes d'impératifs que distinguait Herbert Simon, (A) concernant les objectifs à atteindre, (B) concernant le plan d'action à construire et (C) concernant la méthode générale de telles constructions[168].

La ratiocination est l'investigation (ou la partie d'une investigation) qui procède par l'observation scrupuleuse des règles d'une procédure déjà disponible – et qui évite ainsi les efforts de la recherche initiale. Sa forme emblématique et extrême (par l'exactitude et la fiabilité des règles) est le calcul arithmétique[169].

164. Voir plus haut, § 163, § 173.
165. Voir plus haut, § 173.
166. Voir plus haut, § 134.
167. Voir plus haut, § 148, § 174 et § 178.
168. Voir plus haut, § 65 et § 173.
169. Voir plus haut, § 173.

§ 189. La résolution pratique et technique de problèmes

La plupart des actions sont non réfléchies et « habituelles ». Il ne faut pas comprendre par là qu'elles sont nécessairement stéréotypées et « mécaniques », mais plutôt qu'elles relèvent d'un savoir-faire déjà incorporé par l'agent, qui *sait* d'emblée quoi faire face à une situation, même si c'est la première fois qu'il la rencontre. Il y trouve suffisamment de marques familières sur lesquelles prendre appui et procéder selon son intention initiale[170].

Une telle manière habituelle d'agir est celle de la « vie de tous les jours » et elle s'enracine dans la profondeur de tous nos automatismes physiologiques, innés ou acquis – marcher, respirer, etc. Elle est également à l'œuvre dans nos pratiques, au sens de MacIntyre[171] ; elle résulte alors en général d'un apprentissage et d'une discipline ; elle peut concerner aussi bien des actions physiques qu'intellectuelles : comme on lit naturellement des mots sur une page, un joueur d'échecs lit naturellement une situation sur un échiquier, et dans les deux cas ces actes de lecture évoquent des significations, des émotions et des possibilités d'action. Toute profession ou occupation requiert une certaine intelligence des situations, spécifique à cette pratique ; de manière générique, c'est également de cette manière d'agir que relève le sens pratique de Pierre Bourdieu, avec ses dimensions sociales[172].

Lorsqu'une situation problématique apparaît, c'est-à-dire une rupture dans le flux de cette expérience intelligente de la situation, le recours à l'action silencieuse de l'entendement, qui considère les significations (actions possibles, états possibles de l'expérience) dans leurs simples relations symboliques, permet à l'agent de s'extraire de ses habitudes et de porter un premier regard réflexif sur ses propres pratiques[173]. C'est ce qu'on appelle la délibération, et le savoir pratique qui en résulte (par exemple, qu'il est en général bon de consulter la météo avant de négocier des denrées agricoles) est déjà au dessus du simple savoir-faire, parce qu'il est discursif. Cet exemple ne doit pas faire croire qu'on est limité ici à la considération de fins dites *utiles*, orientées vers une certaine catégorie de besoins « concrets » comme la nourriture, l'enrichissement, etc. Dans une perspective comme celle de Dewey, toute perplexité, comme celle que pourrait susciter une comète traversant le ciel pour ne plus revenir, toute fêlure dans le monde de l'expérience, qui est immé-

170. Voir plus haut, § 167.
171. Voir plus haut, § 74.
172. Voir plus haut, § 75.
173. Voir plus haut, § 168.

diatement pour nous un monde d'inférences, peut devenir un problème qui exige résolution. Comme nous l'avons dit, toute manipulation de nos représentations est déjà une action, même si elle est apparement purement « mentale ».

Lorsqu'on commence à *rendre public* ce savoir pratique en énonçant des règles générales, mais surtout en *organisant* la situation afin de faciliter l'action, en la dotant d'outils, de moyens cohérents, de protocoles d'interaction entre les personnes, en organisant des processus de maintenance et de supervision de ces environnements, alors on peut parler de *savoir technique*, qui est un savoir pratique *dans* cet environnement[174]. Il est à noter qu'il n'y a pas lieu ici de distinguer fortement entre techniques et institutions ; les premières ont plutôt trait aux interactions avec les choses, et les secondes entre les personnes ; mais elles relèvent d'une même logique, celle d'organiser des situations afin de faciliter la réalisation des intentions des personnes. On associe souvent ces environnements aux règles et aux procédures ; cependant ces savoirs discursifs n'épuisent pas la totalité du savoir technique et institutionnel des agents ; comme nous l'avons vu[175], les règles ne visent pas tant à décrire les techniques et les institutions qu'à les compléter lorsque les contraintes inscrites dans ces environnements sont jugées insuffisantes pour guider les agents. Tous ces savoirs restent donc éminemment contextuels et particuliers.

Les personnes qui agissent au sein de ces environnements peuvent les enrichir, parfois de manière implicite, en favorisant des « pratiques » usuelles ou coutumières, et parfois de manière explicite, soit en y laissant des marques additionnelles – ainsi des gabarits que conserverait un menuisier après une commande qu'il juge devenir régulière – soit en énonçant des règles complémentaires – ainsi la jurisprudence en droit.

Afin d'être proprement incorporés à un corpus technique ou institutionnel, un artefact ou une procédure sont l'objet de vérifications plus fortes que la solution simple d'une investigation particulière, puisqu'ils doivent, contrairement à cette dernière, servir dans un grand nombre de cas, ayant de nombreuses configurations. Ces vérifications peuvent être faites *a posteriori*, c'est-à-dire par la sanction de leur usage même qui, petit à petit, les ajuste et les polit aux différentes intentions et situations qui les mobilisent, ou *a priori*, c'est-à-dire par un examen réfléchi d'ordre supérieur qui tente d'envisager par avance la procédure ou l'artefact dans ces différentes configurations.

174. Voir plus haut, section 4.4.
175. Voir plus haut, section 5.1.

Un tel exercice suppose qu'on soit capable de se représenter les problèmes auxquels l'artefact ou la procédure doivent répondre avec une certaine généralité. Cela requiert à son tour l'effort de cette réflexion disjonctive concernant les configurations possibles de la situation, que nous évoquions tout à l'heure. À moins qu'elle n'ait affaire à de pures opérations, cette réflexion est contrainte par les systèmes de significations dont on dispose. Par exemple les ingénieurs des ponts au XVIII^e siècle ne concevaient pas qu'un jour des trains pourraient les traverser.

Dans de tels exercices on touche déjà à ce que nous avons appelé une *investigation globale*, puisqu'on tente de penser l'insertion d'une procédure ou d'un artefact nouveaux dans un ensemble avec lequel ils doivent entrer en cohérence : par exemple un pont dans un réseau de routes, qui prend en compte le trafic, l'hydrométrie, les usages, etc. Cette investigation globale reconfigure le problème particulier (la construction du pont) en le positionnant aux croisements d'un réseau de problèmes apparentés avec lesquels il doit être solidaire[176].

Lorsque c'est l'ensemble technique ou institutionnel tout entier qui est pris comme objet de l'investigation – comme par exemple lorsqu'il s'agit de l'urbanisme d'une ville, d'un réseau de fortifications, ou d'un réseau de routes, pour rester dans le domaine des infrastructures traditionnelles, et qu'on se propose de les reconcevoir complètement, on est tout à fait dans l'investigation globale. Ce terme est justifié parce que d'emblée il est posé qu'il ne s'agit pas de résoudre un problème local, mais de se doter d'une architecture robuste afin que par la suite la résolution de tout un ensemble de problèmes particuliers s'en trouvent facilitée. Cela n'empêche pas que, dans les faits, ces investigations globales se traitent comme des séries de problèmes particuliers qu'il s'agit de résoudre l'un après l'autre. Ces problèmes d'*infra*-structure sont distincts, en général, des problèmes cibles qu'il s'agit de résoudre.

§ 190. Investigation globale et systèmes

Dans la description donnée ci-dessus de la constitution des ensembles techniques et institutionnels, nous avons évité l'emploi du terme *système* ; la plupart d'entre eux n'ont pas une visée systématique, même si toute nouvelle règle qui leur est incorporée est vérifiée soigneusement. L'atelier d'un artisan est rarement pensé de manière systématique, non plus qu'une organisation : de nouveaux événements surviennent régulièrement qui les remettent en cause et les oblige à s'adapter et à

176. Voir plus haut, § 183.

se reconfigurer. Il est donc crucial que ces environnements laissent explicitement la place, les uns à l'intelligence des situations techniques particulières[177], les autres à la négociation entre les personnes. Il peut même sembler *absurde*, à première vue, de chercher une telle systématicité dans ce type de lieux[178].

C'est donc d'abord dans le règne des significations qu'une telle forme d'investigation globale est d'abord possible à proprement parler, lorsqu'il s'agit d'opérer un retour réflexif sur le savoir d'arrière-plan que mobilisent les investigations particulières du champ ou de la discipline qu'il s'agit de réformer.

Une théorie est *systématique* en ce qu'elle prétend organiser et contrôler de manière exacte et exhaustive les significations (ou encore les règles) dont elle a la charge[179]. Elle doit d'abord s'assurer qu'elle permet d'exprimer une classe donnée de problèmes c'est-à-dire sans que ceux-ci contiennent un seul terme qui lui soit extérieur : ce que nous avons appelé l'*exactitude*[180]. Elle doit fournir les règles pour les résoudre, bien qu'il ne soit pas obligatoire qu'elle soit complète, au sens mathématique, c'est-à-dire qu'elle garantisse qu'ils soient tous solvables. Elle doit enfin donner les moyens de vérifier la fiabilité exhaustive des solutions proposées sur l'espace de variabilité du problème (si celui-ci est fini, une telle possibilité ne pose pas de problème). Ce faisant, elle cherche à la fois à *décontextualiser* les problèmes et à les *factoriser* – et ce travail la conduit à se réexprimer en fonction d'invariants toujours plus élémentaires[181].

L'effort de systématisation peut aller à son terme lorsque ses significations élémentaires sont entièrement *procédurales*, c'est-à-dire qu'elles se réfèrent à de pures relations entre opérations dans l'expérience – par opposition aux significations matérielles qui dénotent l'attente qu'une certaine expérience vécue résulte d'une opération[182]. Les systèmes mathématiques portent ainsi seulement sur des transformations d'opérations et non sur des expériences complètes ; par exemple l'arithmétique étudie les rapports entre les opérations sur les nombres, qui eux-mêmes désignent des *actes* de compter[183].

La théorie des types de Martin-Löf permet d'exprimer de manière générique

177. Voir plus haut, section 5.1.
178. Voir plus haut, § 108.
179. Voir plus haut, § 180.
180. Voir plus haut, § 160.
181. Voir plus haut, § 181.
182. Voir plus haut, § 172.
183. Voir plus haut, section 10.3.

ces systèmes de transformations opérationnelles[184]. Un type élémentaire est un ensemble de règles décrivant des structures inductives de composition possible d'opérations ainsi que les relations valant entre leurs résultats – par exemple, qu'il est équivalent de compter d'abord jusqu'à 3, puis jusqu'à 4, ou de compter d'une traite jusqu'à 7. Les relations qui sont établies entre types eux-mêmes dénotent l'équivalence de telles structures opérationnelles tout entières – comme par exemple que l'opération de doubler un nombre implique que la vérification de la parité du résultat sera valide. Quels que soit les niveaux élevés d'abstraction qu'on peut atteindre ici, il n'y est jamais question que de cohérence entre opérations possibles, telles qu'elles sont définies les unes à partir des autres.

Une théorie *de l'expérience*, quant à elle, c'est-à-dire qui porte sur des phénomènes empiriques – par exemple la mécanique, ou la biologie, peut s'appuyer sur les mathématiques dans la mesure où elle parvient à définir des observables, ainsi que les protocoles de mesure de ces observables, qui deviennent l'interface de la théorie avec l'expérience ; une fois qu'ils lui sont donnés, elle peut les manipuler *systématiquement* dans le cadre mathématique ou logique qu'elle s'est donnée, dont les règles sont entièrement procédurales.

Dans l'investigation théorique, il y a ainsi un appauvrissement de l'expérience – ce qui ne peut être systématisé est exclu du système[185] – mais également un enrichissement, car les inférences entre significations sont combinées entre elles et comparées exhaustivement, si bien que sont identifiés de nouveaux problèmes, de nouvelles pratiques opératoires ainsi que des nouveaux moyens d'action, qui à leur tour instillent d'autres progrès théoriques. C'est ainsi que toute investigation globale significative est toujours un *programme* de recherche dont l'achèvement est régulièrement repoussé[186].

Les méthodes industrielles et bureaucratiques sont l'application de cette approche systématique aux ensembles techniques et institutionnels ; afin d'assurer l'exactitude et l'exhaustivité des procédures, il est nécessaire de concevoir un lieu clos, soigneusement séparé du monde environnant[187], régi lui-même par des règles exactes, et dont les artefacts sont eux-mêmes idéalement le produit d'autres procédures exactes. Comme le dit Marx[188], il fallait que les machines elles-mêmes

184. Voir plus haut, section 9.4.
185. Voir plus haut, § 181.
186. Voir plus haut, § 182.
187. Voir plus haut, § 86, § 109, § 185.
188. Voir plus haut, p. 719.

commencent à se produire elles-même afin que l'industrie se lève et marche de ses propres jambes.

Les systèmes techniques et institutionnels – ce que nous appelons les systèmes *objectifs* – obéissent donc à la même logique que les systèmes théoriques, mais ils guident et assurent la délibération et l'action en organisant directement les objets et les ressources de l'expérience, qui sont des signes naturels, directement porteurs d'inférences. En tant qu'ils visent la systématicité, ils transforment les écologies cognitives naturelles et traditionnelles des personnes humaines de la même manière que les systèmes théoriques transforment leur langage naturel et le sens commun. C'est là que peut être aperçue la proximité intellectuelle de la démarche industrielle d'organisation des pratiques et de la démarche scientifique : toutes deux visent à contrôler la délibération et le jugement et à en garantir l'effectivité, en leur proposant des espaces de problème et des systèmes de règles préparés à l'avance, génératifs de l'ensemble du domaine visé[189].

§ 191. La résolution systématique de problème

Ce cadre étant posé, il est à présent possible de comprendre ce qu'est la résolution *systématique* de problème, et en quoi elle s'oppose à l'investigation *pratique* ou *technique* simple. On peut en fait comprendre cette expression en plusieurs sens.

Dans le sens le plus général et le plus lâche, selon lequel Dewey interprèterait sans doute une telle expression, il s'agit tout simplement d'expliciter la solution candidate dans les termes du système (sa *mise en procédure*) et de la vérifier selon les normes de correction qu'il impose. La démarche scientifique, en particulier, vise à mettre à disposition de l'investigation des systèmes de significations contrôlées dans leur contenu propre et dans leurs relations, de manière aussi complète que possible. La solution d'une investigation s'exprimant en ces termes bénéficie par là-même de tous ces « travaux de terrassement » préliminaires et sa propre vérification s'en trouve par là simplifiée. Cependant, une telle procédure n'est pas pour autant assurée d'être exhaustive[190].

Cette exhaustivité est cependant essentielle au sens intuitif dans lequel nous avons d'abord employé le terme de *systématicité*, dans le cadre de notre analyse

189. Voir plus haut, § 184.
190. Voir plus haut, § 181.

de la programmation[191] : il faut que, *quels que soient les paramètres du problème*, la *solution* retenue ne rencontre pas d'exception, d'impossibilité ou de flou, c'est-à-dire qu'elle offre une forme de *garantie* de solution pour une classe de problèmes *exactement* définie.

À lire Dewey, il semble souvent qu'il a à l'esprit des systèmes taxonomiques (arborescents ou combinatoires). Dans de tels modèles, la preuve d'exhaustivité n'a souvent pas d'autre option que de parcourir l'ensemble des branches ou combinaisons possibles une à une. Il s'agit d'une ratiocination pure, qui est souvent qualifiée de *force brute* (*brute force*)[192] par les informaticiens. Traditionnellement regardée comme la pire des méthodes (parce que la plus inefficace) la puissance informatique la rend praticable pour certains problèmes où toute autre méthode échoue. Ainsi le théorème des quatre couleurs, que nous examinons plus loin[193], fut démontré grâce à l'examen exhaustif d'un millier et demi de configurations possibles de l'espace de problème. Les problèmes qui peuvent être réduits au problème de satisfiabilité booléenne (SAT)* sont également aujourd'hui candidats à ce type d'approches, grâce aux algorithmes puissants qui permettent de les résoudre à un facteur d'incertitude très faible près[194].

Si les systèmes ne pouvaient cependant prendre qu'une telle forme, leur valeur pour la résolution de problème serait limitée ; ils ne pourraient exprimer de nouveaux problèmes qu'en se prolongeant eux-mêmes en de nouvelles branches ou en de nouveaux termes combinatoires, des opérations problématiques qu'il faudrait à leur tour vérifier. On attend au contraire d'un système qu'il reste stable tout en permettant à de nouveaux problèmes d'être décrits et résolus en son sein, et cela de manière indéfinie[195].

Nous avons identifié trois principes[196] auxquels devaient se conformer les règles d'un système afin que celui ait de telles propriétés génératives. Il faut d'abord que les règles puissent se composer entre elles de manière indéfinie (principe de composition), que chacune porte sur un nombre fini de conditions de la situation (principe de localité) et enfin que la configuration de ces

191. Voir plus haut, § 29.
192. Dans cette expression anglo-saxonne, *brute* doit être entendue dans le sens d'une *bête brute*, d'une attitude fruste, sans intelligence.
193. Voir plus loin, § 215.
194. (Heule et Kullmann 2017).
195. Voir plus haut, § 116.
196. Voir plus haut, § 160.

conditions puisse elle-même être engendrée selon des règles (principe d'induction). *« Composer, selon les règles du système, la transformation d'une situation, caractérisée par telles règles, en une situation caractérisée par telles autres »* : telle est la forme générale que peut prendre un problème décrit dans un tel système. Dans ce cas, sa résolution peut prendre appui sur les propriétés structurelles du système de paramètres pour définir la solution et vérifier sa correction, en évitant le parcours exhaustif des situations à traiter.

C'est une telle idée que la théorie des types propose de généraliser. Dans ce cadre de réflexion, tout problème est lui-même un type, qui doit être défini exclusivement à partir des types disponibles dans le système considéré, certains pouvant servir de paramètres. Ces types possèdent une règle d'induction qui explique comment parcourir leurs éléments et par là définir de nouveaux objets qui les prendraient comme paramètres : il suffit en effet, à partir d'un objet déjà défini pour un paramètre donné, de savoir comment définir le suivant[197].

Définir la procédure de résolution d'un problème, c'est montrer comment construire de tels objets d'une telle manière inductive ; son exhaustivité est alors garantie, du moment que cette construction est prouvée correcte.

Il est intéressant de réfléchir aux différences et similitudes qui existent entre ces deux méthodes (« force brute » *vs* méthode inductive) pour prouver l'exhaustivité d'une résolution de problème. La première semble n'exiger que la discipline de la ratiocination, et la seconde, seulement l'intelligence de la conception. La première semble être de nature *extensive* – exécutant *effectivement* la procédure sur *tous* ses paramètres (jusque dans une certaine limite s'ils sont infinis), la seconde de nature *intensive*, donnant la raison même pour laquelle cette exhaustivité peut être garantie, *sans effectuer la vérification* elle-même.

Cependant, ces différences nous semblent superficielles, si on les rapporte aux autres manières de résoudre des problèmes que nous avons décrites plus haut. D'une part, la « force brute » des algorithmes exhaustifs requiert que leurs ratiocinations soient préparées avec attention ; l'essentiel du travail sur le théorème des quatre couleurs se situe *en amont* de la ratiocination automatisée qui en a achevé la démonstration. D'autre part, les « raisons intensives » de l'exhaustivité d'une procédure, que permet d'exhiber le cadre de la théorie des types, n'ont plus rien à voir avec les propriétés essentielles ou sémantiques qu'on associe en général aux définitions intensives. Il n'est question ici que des règles opératoires à appliquer

197. Voir plus haut, § 161.

pour parcourir de manière extensive une collection d'éléments.

Ce qui est donc essentiel dans les systèmes purs exprimés en théorie des types, c'est que toute solution d'un problème ne peut avoir que deux formes fondamentales : soit l'exécution fruste d'une procédure disponible, soit la recherche intelligente d'une telle procédure[198]. Entre ces deux formes, il n'y a plus aucune place pour l'intelligence des situations dont nous parlions plus haut. Ces deux formes expriment parfaitement la division stricte du travail entre conception et exécution qui commence à se mettre en place aux Temps modernes, et dont l'exécution mécanique est l'emblème[199]. L'idée directrice de la systématicité est de supprimer *tout besoin de futures investigations* sur une même classe de problème, en les remplaçant par des ratiocinations où tout besoin de jugement est définitivement aboli.

Au terme de cette synthèse, a-t-on répondu aux deux questions directrices (sous toutes leurs formes successives) qui ont animé notre réflexion épistémologique ?

Concernant la première, nous avons montré l'articulation intime de la connaissance de *ce qu'il faut faire* et de *ce qui est* dans les significations (ou règles) qui sont le matériau de toute investigation. Situation et action sont d'abord des catégories logiques propres à l'investigation et tout jugement, lorsqu'il vise la certitude, doit expliciter sa propre preuve, c'est-à-dire la série d'inférences qui conduisent à une telle *décision*, dans l'expérience présente.

Concernant la seconde, nous avons montré comment les procédures étaient, d'une manière générale, capables d'augmenter la connaissance. En tant qu'elles sont la récapitulation d'investigations, ainsi mises à disposition de futures investigations afin de les abréger (*via* la ratiocination) elles sont un moyen essentiel de son accroissement. Ce mouvement d'accumulation est de même nature que celui qui préside à la construction de *systèmes de connaissance* (investigation globale), qui rendent possible la résolution systématique de problème. Nous avons tenté de montrer le caractère explosif, pour l'accumulation du savoir, de cette coopération entre investigations globales et locales *systématiques*.

Cependant, cette réponse à la deuxième question est incomplète. D'abord, nous n'avons pas encore expliqué en quoi des machines pouvaient se rendre utiles dans cette économie de la connaissance. Ensuite (et cette seconde question permet sans doute de répondre à la première) l'idée que la mise en procédure et la systématisation permet seulement d'*économiser du temps* d'investigation (que ce soit en re-

198. Voir plus haut, § 147.
199. Voir plus haut, § 118.

cherche, en vérification, en transmission, etc.) est insatisfaisante. C'est sans doute ce à quoi elle était limitée avant l'arrivée des machines, et explique que le concept de procédure se soit si peu développé *en tant que tel* jusque là, mais notre étude préliminaires des avantages d'effectivité[200] procurés par l'industrialisation et l'informatisation suggère un rôle bien plus large des procédures.

La résolution systématique de problème apparaît ainsi intimement liée au projet de ratiocinations *automatisées* ou *mécaniques* desquelles tout besoin de jugement a disparu. Nous sommes donc à présent prêts à examiner la signification de ces termes : *automatisé*, *mécanique*, à comprendre *ce qu'est une machine* et quelle forme de connaissance est la programmation.

200. Voir plus haut, section 5.4.

Cinquième partie

Machines et systèmes

Chapitre 11

La programmation de machines

§ 192. Introduction

Suite à notre analyse du modèle contractuel de la programmation, à la fin de la première partie, nous nous sommes demandé si cette activité pouvait prétendre être la forme achevée de la raison pratique, ainsi que le suggérait Herbert Simon lorsqu'il prenait la programmation pour modèle de ses « sciences de l'artificiel ». Nous avons répondu par la négative, en montrant au contraire que les procédures informatiques ne pouvaient nullement être vues comme des transpositions ou des précisions des plans par lesquels les personnes humaines fixaient en général le cours de leurs actions, ou des méthodes au travers desquelles ils explicitaient et transmettaient leurs savoir-faire. Proches en cela des méthodes industrielles, les procédures informatiques font souvent *violence* à la rationalité pratique humaine et à ses écologies cognitives. Nous avons alors fait l'hypothèse qu'informatique et industrie procédaient d'une même attitude, celle de la *résolution systématique de problème*.

Les deux parties précédentes ont permis de dégager un cadre épistémologique minimal autour de cette notion. Nous avons placé celle-ci dans la perspective du débat sur le temps long entre Aristote et Kant concernant les rapports entre raison pratique et raison théorique, nous l'avons rendue formellement exacte par la théorie constructive des types, et nous l'avons enfin fondée dans la théorie de la connaissance de Dewey. Nous pouvons ainsi revenir, dans cette dernière partie, aux deux questions générales qui guident l'ensemble de ce travail :

1. Qu'est-ce que programmer ? et quelle forme de logique et de connaissance est engagée dans cette activité ?

2. Comment expliquer la capacité qu'a la programmation des machines à se rendre essentielle à presque toute activité rationnelle humaine, qu'il s'agisse de techniques, de sciences, d'administration, de communication, etc. ?

Concernant la première question, la partie précédente nous suggère d'identifier la programmation avec la *mise en procédure d'une investigation*, c'est-à-dire à sa *récapitulation procédurale et systématique*, telle qu'on peut la caractériser dans l'épistémologie de John Dewey[1].

Une telle suggestion est tentante, car elle expliquerait de manière frappante pourquoi la programmation s'infiltre si bien dans tous les domaines de la pensée et de l'action humaine : ce serait simplement que tous ont affaire à l'investigation. On peut « mettre en procédure » des preuves mathématiques aussi bien que des procédures juridiques ou administratives, des stratégies militaires, des méthodes de diagnostic médical, la conduite d'un avion ou celle d'un engin agricole. La programmation serait alors possible dans la mesure proportionnelle où les significations du domaine en question se prêteraient à la systématisation, c'est-à-dire à leur reformulation exacte et au contrôle exhaustif de leurs relations.

Par ailleurs, une telle identification permettrait d'identifier également l'informatique avec la *science des opérations*, telle que nous l'avons caractérisée suite à notre interprétation des travaux de Per Martin-Löf[2]. Enfin, tout cela entrerait en résonance avec notre interprétation aristotélicienne de la technique. Si on conçoit en effet la délibération aristotélicienne comme une investigation (*ζήτησις*), alors la mise en procédure est le mouvement même d'universalisation de la délibération – son moment technique, ainsi que nous l'avions caractérisé alors[3]. Mettre en procédure, dans ce sens, est le geste technique par excellence.

Cependant, de telles identifications présentent plusieurs difficultés.

D'abord, dans l'épistémologie de Dewey, ce sont les mathématiques tout entières qui étudient les moyens procéduraux, et qui donc peuvent être également qualifiées de sciences des opérations, ce qui les confondrait avec l'informatique. Par ailleurs, le rapport de la programmation à la résolution de problème n'est pas univoque. Il est possible que la programmation contribue seulement de manière partielle et provisoire à l'investigation, et non qu'elle la récapitule, par exemple lorsqu'on veut explorer systématiquement les conséquences d'une hypothèse. Ici

1. Voir plus haut, § 173 et § 188.
2. Voir plus haut, section 9.4.
3. Voir plus haut, § 99.

la programmation est « micro-locale » (si par *investigation locale* nous entendons la résolution d'un problème bien précis[4]). En sens inverse, certains projets de programmation visent à créer des systèmes génériques de résolution de problèmes : ainsi lorsqu'on programme un langage de programmation, ou même lorsqu'on transcrit une théorie mathématique dans un assistant de preuve comme Coq afin de pouvoir ultérieurement y établir des preuves. Ici, la programmation ressemble bien plus à ce que nous avons appelé des *investigations globales*, dont font partie les recherches théoriques[5].

Ensuite (et sans doute pour les mêmes raisons) dans ces identifications le rapport entre la programmation et les machines est escamoté, comme s'il n'était pas essentiel. Or nous avions jugé centrale la notion de *machine* dès notre première définition de la programmation[6] et son importance n'a depuis cessé de croître dans notre réflexion ; en particulier, elle est centrale dans l'interprétation contractuelle que nous avons donnée de la programmation[7].

De ce fait – et cela nous amène à notre seconde question directrice – il semble douteux que de telles identifications puissent suffire à expliquer l'effectivité de l'informatique, puisqu'il n'y est nulle part question d'automatisation. On conçoit peut-être que mettre en procédure des investigations permettrait, dans certains cas de figure, de gagner du temps sur des investigations futures, mais on a vu que des exécutants humains seraient la plupart du temps mal à l'aise dans l'exécution de procédures prenant la forme stricte des programmes informatiques. On semble alors être réduit à constater que l'automatisation des raisonnements et des plans d'actions est un accident heureux qui récompense une pensée exacte et exhaustive, avec l'aide d'une certaine ingéniosité technique qui découvre comment la transcrire dans les circuits de machines électroniques.

Il y a cependant, dans les suggestions évoquées, une intuition qui nous suit depuis quelques temps, à savoir que la programmation a affaire essentiellement à l'investigation, et notamment à sa reconstruction procédurale. Il nous faut donc surmonter les difficultés mentionnées et parvenir à préciser cette intuition pour la rapprocher de notre interprétation contractuelle de la programmation.

L'idée directrice qui va guider cette partie est que les machines, insérées dans une situation problématique, servent à y rendre *directement effectives* ces procé-

4. Voir plus haut, § 142.
5. Voir plus haut, § 190.
6. Voir plus haut, § 40.
7. Voir plus haut, section 3.3.

dures systématiques établies par l'investigation. En d'autres termes, un agent « mécanique » effectue automatiquement dans la situation problématique les règles que l'investigation établit comme devant la résoudre.

On voit bien comment cela est possible lorsque l'investigation porte, justement, sur un mécanisme, et que la procédure résultante est un raisonnement décrivant son fonctionnement – par exemple, que le déplacement du pignon sur la crémaillère fera tourner l'arbre de la poulie, transformant ainsi un mouvement rectiligne en mouvement circulaire – mais ici il est question de rendre effectives les récapitulations de *toutes* sortes d'investigations, du moment qu'elles sont systématiques. Cela requiert donc de comprendre *comment il est possible* que des machines puissent exécuter des procédures récapitulant des investigations *avant tout humaines*, c'est-à-dire exprimées dans des significations n'ayant de sens que pour des personnes. Ici, l'exigence de systématicité des procédures semble centrale, une machine ne pouvant gérer ni des significations approximatives ou ambiguës, ni des exceptions procédurales.

Autrement dit, la complémentarité de ces deux aspects de la programmation – insertion d'une machine dans une situation problématique, reconstruction systématique d'une investigation – semble donc bien revenir, encore une fois, à celle qu'il doit y avoir entre les notions de *systématicité* et de *machine*, et que nous devons terminer donc d'élucider.

Concernant notre seconde question directrice, il nous faut réciproquement tâcher de comprendre pourquoi les gains d'effectivité qu'apporte l'informatique ne sont pas les « accidents heureux » d'une approche systématique de la connaissance, mais des parties de sa visée essentielle – de rendre effectif le rapport de la connaissance au réel.

La difficulté d'une telle entreprise est qu'on ne parle pas ici d'un domaine bien délimité de connaissance – par exemple des sciences, ou des techniques, ou de certaines pratiques humaines – mais de tout cela en même temps, si bien que le projet même de chercher une explication unique à une telle effectivité peut sembler douteux, comme l'avance par exemple l'historien Mahoney[8]. C'est bien pourtant ce qui nous guide ici – chercher à comprendre *ce qu'il y a de commun* dans toutes les entreprises de connaissance qui se prêtent à une approche systématique, et qui « donne prise » à l'informatique.

De multiples questions sont donc entremêlées dans cette dernière partie qui doit

8. (Mahoney 2011, p. 61).

nous permettre de conclure notre travail ; elles seront résolues progressivement au cours de ces deux chapitres. Nous synthétiserons donc toutes ces réponses dans le dernier développement § 235 de cette partie (et non à la fin de ce chapitre).

Nous commençons (section 11.1) par établir, grâce à l'analyse du concept de *procédure effective*, pourquoi il est raisonnable et tentant d'identifier l'informatique à la *science des opérations*. Cela exige cependant de la distinguer des mathématiques, qui peuvent également prétendre à ce titre. Nous discutons en particulier les thèses qu'elle serait une *mathématique des algorithmes* (Knuth) ou une *pragmatique des mathématiques* (Gorn et Caracciolo). Une fois affermie cette base conceptuelle, nous pouvons l'utiliser pour développer une esquisse d'épistémologie positive de l'informatique, qui rend compte de ses principaux domaines d'étude (algorithmique et méthodes formelles) et pratique (heuristique et conception (*design*)) (section 11.2).

Comment cependant peut-on développer une telle épistémologie de l'informatique sans sembler le moins du monde se référer aux machines qui la rendent possible ? Rien n'exige en effet de l'exécutant en question qu'il soit une machine, et d'ailleurs : comment cela serait-il seulement possible, si les procédures sont les récapitulations d'investigations avant tout *humaines* ? Et que veut dire, bizarrement, l'exigence répétée des théoriciens de la calculabilité que l'exécutant se comporte *mécaniquement*, alors que c'est la machine qui semble ici se rendre capable d'imiter la réflexion humaine ? Ces difficultés nous amènent à renverser le regard, à voir *ce qui est déjà* mathématique dans l'idée de *machine*, et à considérer celle-ci comme une catégorie de l'investigation, c'est-à-dire comme une certaine visée descriptive des phénomènes de l'expérience (section 11.3).

En particulier, nous montrons le rôle spécial que jouent certaines machines élémentaires, les *contrôles de transmission*, dans l'établissement de micro-espaces de systématicité, dont nous montrons la double effectivité, industrielle d'abord, en ce qu'elles sont le centre de contrôle des machines, et logique ensuite, en ce que certaines d'entre elles (les machines universelles) ont la capacité de représenter tout système fini de règles et donc d'automatiser ce que nous avons appelé la *ratiocination*[9], c'est-à-dire l'exécution scrupuleuse de procédures (section 11.4).

Ce privilège des machines à contrôle de transmission est si frappant qu'il semble justifier l'hypothèse pan-computationnelle, qui affirme que le traitement de l'information serait la fabrique même du réel. L'examen de cette thèse

9. Voir plus haut, § 173, et § 188.

métaphysique nous permet d'affiner notre concept de *mécanisme* et de rejeter le caractère exceptionnel des machines à contrôle de transmission, tout en leur reconnaissant un certain privilège épistémologique (section 11.5).

Cette hypothèse étant écartée, nous pouvons commencer notre examen des raisons essentielles de l'effectivité des ordinateurs. La dernière section de ce chapitre est consacrée à l'étude de ce qu'apporte la programmation aux investigations systématiques elles-mêmes, et en particulier à celles des mathématiques, des sciences et de l'ingénierie, à l'aide de quelques exemples. Nous montrons qu'il serait réducteur d'y voir un simple gain de productivité sur les ratiocinations (ici, principalement des calculs) que requièrent fréquemment ces investigations. L'expression computationnelle de théories réalise l'exigence de contrôle et d'exactitude des significations qui est au cœur du projet systématique des sciences modernes. L'utilisation de la programmation pour *explorer* des espaces de systématicité (par exemple, comme nous l'avons vu plus haut, les conséquences d'une hypothèse) permet d'étendre le champ de la systématicité, notamment en direction de ce qu'on appelle aujourd'hui les *systèmes complexes* (section 11.6).

Le dernier chapitre, enfin, explore l'effectivité des ordinateurs hors du champ de l'investigation, au sein même des pratiques humaines. Ici, nous développons l'idée centrale que les ordinateurs sont avant tout des machines capables de rendre des règles effectives au sein même de la situation dans laquelle elles sont insérées. Cela est possible parce que les machines, de par leur définition même, sont des objets de l'expérience capables d'interaction. C'est par là qu'elles peuvent transformer les pratiques humaines dans les deux dimensions, technique et institutionnelle, où celles-ci se codifient en règles[10]. C'est grâce à ces règles, qu'elles soient ancrées dans les régularités de la nature ou les stipulations des conventions humaines, que des pratiques peuvent être automatisées pour les bénéfices déjà mis en évidence[11]. Mais il y a bien plus que cela, car il n'est pas toujours question d'automatisation mais plutôt d'encadrement des pratiques dans le monde économique, social, ou politique. Ici la programmation des règles techniques et institutionnelles leur permet de *s'appliquer automatiquement*, exerçant ainsi une contrainte directe sur la situation et sur les agents humains. Surtout, elle permet d'une part d'affiner ces règles indéfiniment et d'autre part de les rendre dynamiques, c'est-à-dire sensibles à leur contexte immédiat. En d'autres termes, la programmation n'est plus

10. Voir plus haut, § 74.
11. Voir plus haut, section 5.4.

seulement ici programmation de machines pour l'automatisation, mais également de *règles* où les deux autres classes d'avantages de l'informatisation, la contrainte exercée sur la situation (et sur les agents humains), ainsi que la traçabilité des interactions, jouent à plein.

La réflexion de cette dernière partie s'enracine dans les conclusions et les concepts épistémologiques développés dans les trois parties précédentes. Aussi y faisons-nous des renvois fréquents ; ils ne se substituent cependant pas à la bonne intelligence des thèses récapitulées en section 10.6, qui devrait grandement faciliter la lecture.

Deux notes terminologiques sont également utiles. Par *situation*, on le sait[12], nous désignons la portion du monde au sein de laquelle l'investigateur reconnaît son problème et l'y découpe. Nous empruntons à l'informatique son emploi du terme *environnement* pour désigner la situation, en tant qu'elle est préparée techniquement par l'investigateur afin d'assurer le succès d'un processus qu'il a conçu pour résoudre son problème. Il s'agit donc d'un concept relatif, et deux processus interactifs ont donc des environnements distincts au sein d'une même situation, *a minima* parce que chacun se trouve dans l'environnement de l'autre.

Par ailleurs, nous emploierons souvent les termes *gain, avantage, bénéfice* de l'informatisation pour mesurer l'accroissement d'effectivité apportée par les ordinateurs, en réponse à notre seconde question directrice rappelée au début de cette introduction. Il est entendu que ces termes ne signifient aucunement notre appréciation positive de ces phénomènes (bien au contraire), ni même qu'ils augmenteraient objectivement la puissance des personnes humaines, comme nous l'avions noté en introduction[13].

11.1 Procédure et mathématiques

Dans cette section et la suivante, nous souhaitons montrer pourquoi il est plausible d'identifier la programmation à une forme de *mise en procédure* d'une investigation achevée, telle que nous l'avons définie au chapitre précédent[14].

On voit bien ce qu'il peut y avoir d'intuitif dans cette hypothèse. D'une part,

12. Voir plus haut, § 54.
13. Voir § 3 ainsi que notre analyse du concept d'*avantage* comme *effectivité relative* en introduction de la section 5.4.
14. Voir plus haut, § 173 et § 188.

toute procédure qui résulte d'une investigation, par exemple une preuve, ou un plan d'action, lorsqu'elle est *formalisée*, semble s'apparenter à un programme informatique. Ce rapprochement est explicite en mathématiques constructives[15]. Dans les langages de programmation qui s'en inspirent, comme l'assistant de preuve Coq, prouver un théorème consiste à indiquer à la machine une séquence d'opérations transformant un contexte propositionnel en un autre. Dans un grand projet, les plans d'actions sont aujourd'hui présentés sous la forme de *graphiques de Gantt*, qui organisent les tâches selon leurs interdépendances, à la manière de programmes informatiques. D'autre part, nous avons vu que la programmation était immédiatement comprise par les informaticiens comme une entreprise de résolution de problème[16].

Nous avons déjà évoqué l'intérêt et les difficultés de cette hypothèse en introduction du chapitre, et n'y revenons pas. Deux remarques doivent cependant être ajoutées.

D'abord, on pourrait objecter à cette hypothèse que l'écriture de certains programmes ne procède pas de la résolution de problème, mais d'une activité créative – comme cela a lieu dans l'art numérique. Comme nous l'avons déjà noté[17], répondre à cette objection dépasse le cadre de ce travail, car elle engage la question du rapport entre la création libre et l'investigation. Nous la jugeons très pertinente, et nous pensons en effet qu'il existe des intentionnalités qui ne procèdent pas de la résolution de problème. Cependant l'activité artistique n'a pas affaire à la connaissance telle que nous l'entendons ici, aussi laissons-nous de côté ces possibilités de la programmation.

Ensuite, l'expression *mise en procédure* laisse ouverte la possibilité que la programmation ne soit pas partie intégrante de l'investigation, mais qu'elle intervienne après coup, une fois celle-ci achevée. On peut programmer des procédures industrielles qui ont déjà été définies par une investigation antérieure. Dans ce cas, la programmation est une seconde investigation, dont l'objet est de traduire ces procédures du langage technique de la production industrielle vers un langage informatique. Cela ne veut pas dire que cette seconde investigation ne présentera pas des difficultés propres, et ne remettra pas en cause la spécification initiale. Il est courant que des procédures administratives ou industrielles soient profondé-

15. Voir plus haut, § 148.
16. Voir plus haut, section 3.3.
17. Voir plus haut, § 40.

ment remodelées suite à leur informatisation, même si cela n'était pas envisagé initialement.

Cependant, les entreprises de programmation doivent la plupart du temps être conçues comme des investigations visant l'explicitation d'une procédure permettant la résolution d'un problème paramétrique. Plutôt que d'être un complément *a posteriori* de l'investigation, il s'agit ici de contraindre celle-ci dès l'abord, en exigeant de formuler d'emblée le problème dans un cadre général garantissant que ses résultats puissent être mis en procédure, c'est-à-dire que toute solution particulière (un jugement, une décision, une action à prendre) puisse être analysée comme une fonction de la situation initiale, décrite par des paramètres. Par exemple, la mise en procédure du traitement d'une facture impayée doit être analysée dans le cadre plus général du processus de commande, qui permet de distinguer un simple retard de paiement d'un contentieux, et de prescrire en conséquence des actions différenciées. La mise en procédure, ici, n'est pas seulement un mouvement de généralisation et d'unification des solutions d'investigations particulières, mais une entreprise de systématisation d'un domaine, où toute situation pertinente est cartographiée et trouve sa solution.

Nous commençons par examiner à quelles conditions la reconstruction procédurale d'une investigation pourrait être identifiée à un programme. Pour cela, nous suivons attentivement la caractérisation classique que donnent les informaticiens de la notion de *procédure effective* – sa finitude (§ 193) et son exactitude (§ 194). Cela nous permet d'identifier la *mathématisation* de la procédure comme condition centrale de la programmabilité, et nous amène à explorer les thèses qui affirment que l'informatique théorique est une discipline mathématique (§ 195). Nous étudions en particulier l'idée de Knuth, qu'elle serait la *mathématique des algorithmes*, et celle de Gorn et Caracciolo, qu'elle serait la *pragmatique des mathématiques*. En reformulant ces propositions, nous avançons la thèse que l'informatique serait une mathématique de second ordre, qui concerne la résolution de problème, similaire en cela à la théorie de la preuve. Cependant elle n'a pas que des aspects théoriques. Elle est également une mathématique appliquée, qui élabore des concepts et des théories mathématiques (les langages de programmation) pour résoudre des classes de problèmes dans des champs déterminés (§ 196).

§ 193. La notion de procédure effective (1) : finitude

Mettre en procédure une investigation réussie sert d'abord à éviter de la réitérer dans des situations qu'on considère similaires. Par exemple[18], le problème $P2$ (trouver x tel que son double soit 18) exige d'appliquer pas à pas la procédure $demi(y)$, qui en est la solution générale, pour $y = 18$, c'est-à-dire de ratiociner ou encore ici, plus précisément, de calculer.

En première approche, on peut définir le calcul comme une ratiocination *pure*[19], au sens que l'investigation proprement dite se réduirait à un jugement initial qui reconnaîtrait dans la situation problématique un cas prévu par la procédure, et à un jugement terminal qui évaluerait les résultats de la ratiocination, et déciderait de les appliquer (ou non) à la situation. Entre ces deux jugements, seuls seraient nécessaires les actes de compréhension des instructions de la procédure – d'où également le réquisit de ne pas interagir avec la situation dans ce laps de temps. Cela n'est certainement pas ce qui se passe dans la plupart des ratiocinations, qui impliquent des décisions locales et des interactions avec la situation[20]. C'est justement pour tenter de circonscrire les conditions nécessaires à cette « purification » de la ratiocination que la notion de *procédure effective* se dégage au cours les années 1930.

Comme on l'a vu[21], la notion de *procédure effective* est étroitement associée à celle de programme. Elle se trouve souvent en liminaire des manuels d'informatique théorique, car on la présente comme déterminant l'ensemble des procédures exécutables avec succès par un ordinateur. Aussi de nombreuses définitions utilisent directement ce critère. Cependant, pour nous une procédure est d'abord le résultat d'une investigation, et nous ne souhaitons pas utiliser un concept empirique comme celui d'ordinateur pour la caractériser.

Nous allons donc suivre la définition proposée par Knuth[22], qui a le mérite de s'en passer dans une large mesure. Knuth définit la procédure effective comme « un ensemble fini de règles qui donne une séquence d'opérations pour résoudre un type spécifique de problème ». Il la précise par cinq conditions, que nous présentons dans un ordre légèrement différent du sien. (1) Cette séquence doit se terminer

18. Voir plus haut, § 147.
19. Sur la notion de *ratiocination*, voir plus haut § 173.
20. Voir plus haut, § 173.
21. Voir plus haut, § 41.
22. (Knuth 1997, vol. 1, p. 4-6).

en un nombre fini d'étapes. (2) et (3) Le problème doit avoir des objets bien spécifiés (*specified*) en entrée et en sortie. (4) Chacune des opérations indiquées doit être « spécifiée rigoureusement et sans ambiguïté (*specified rigorously and unambiguously*) » de manière à ce que l'expécutant puisse la « comprendre exactement (*understand exactly*) » (5) et exécutable par lui « exactement et en un temps fini (*exactly and in a finite length of time*) », avec les moyens à sa disposition (Knuth parle ici de papier et crayon). Nous appelons (5a) la condition d'exactitude de l'exécution et (5b) celle de sa terminaison.

Cette définition permet de distinguer les procédures informatiques des procédures en général par deux grands critères, la *finitude* et l'*exactitude*, que nous allons examiner tour à tour.

Nous commençons par les points qui concernent le caractère fini de la procédure. La condition (1) énoncée par Knuth est la plus importante et elle est délicate à interpréter. Il est très facile pour une investigation d'énoncer des préceptes qui pourraient ne pas se terminer, comme *« Attends de recevoir tel message avant de faire ceci ou cela ! »*. Cette procédure est ainsi ouverte à son environnement ; nous reviendrons sur l'idée centrale d'*interactivité* au chapitre suivant. Pour l'instant, on peut considérer que de tels cas de figure sont exclus de la notion de procédure effective.

Même une fois cela précisé, une telle condition de finitude semble trop forte, de l'aveu même de Knuth. Par exemple, une procédure qui indique comment écrire les décimales de π *ne doit pas* se terminer. Knuth dit qu'on doit alors parler de *méthode computationnelle* plutôt que d'algorithme. Ainsi la plupart des procédures mathématiques qui emploient des nombres réels sont en ce sens des méthodes computationnelles, et non des algorithmes, comme la méthode de la bissection pour trouver la racine d'une fonction continue. Ces méthodes cependant sont déterminées pour donner des résultats intéressants lorsqu'elles sont arrêtées dans des conditions correctes : on calcule n décimales de π, ou on trouve la racine avec *telle* approximation. On pourrait aussi bien présenter ces méthodes comme des familles infinies de méthodes finies, paramétrées par le degré de précision souhaité du résultat visé.

Ce qui est donc exclus par cette condition (1), outre les procédures interactives, sont celles qui engageraient l'exécutant dans un travail dont la condition de terminaison pourrait n'être jamais atteinte, sans donner aucune fraction du résultat attendu en cas d'un arrêt impromptu. On peut donner pour exemple une procé-

dure déficiente de sortie d'un labyrinthe, qui entraînerait un parcours circulaire. Une telle procédure peut, dans l'absolu, résulter d'erreurs manifestes au sein de l'investigation, c'est-à-dire d'erreurs de jugement. Cependant le processus même de reconstruction de l'investigation en général suffit à les débusquer. Il est donc bien plus plausible que les erreurs proviennent du processus de reconstruction lui-même, et notamment d'une généralisation défaillante de cas particuliers réussis : une méthode qui réussit dans de nombreuses configurations de labyrinthes, pourrait échouer dans d'autres. C'est donc l'exigence d'une validité *exhaustive* de la procédure sur son espace de paramètres qui apparaît ici en filigrane de cette condition (1).

Les mêmes analyses s'appliquent à la condition 5b), que chaque règle puisse être exécutée en un temps fini, puisque toute règle est elle-même une procédure[23].

Enfin, le préambule de la définition de Knuth exige que la procédure soit exprimée par « un nombre fini de règles ». Cette exigence doit être également précisée, car elle peut sembler redondante avec celles que nous venons de voir. En effet, si la procédure s'exécute en un nombre fini d'étapes, et qu'à chaque étape correspond une seule règle ou un nombre fini de règles, ne peut-on pas conclure que le nombre total de règles utilisées par la procédure est également fini ?

Cela n'est pas nécessairement le cas si le problème est paramétrique : il se pourrait alors qu'un nombre infini de règles soit nécessaire pour définir la procédure sur l'ensemble de ses conditions possibles, même si seulement certaines d'entre elles seraient utilisées dans chaque résolution particulière de problèmes. Un tel cas est en fait courant : ainsi la règle « *Ajouter* n *au résultat* » pourrait être vue comme une famille infinie de règles paramétrées par n. Cela ne pose pas de problème, car une telle famille de règles peut être définie de manière inductive[24], et donc réduite à un nombre fini de règles. Par cette exigence, Knuth sous-entend que seules de telles règles sont autorisées. Ce point est capital. On pourrait en effet disposer d'une famille infinie de règles qui, à chaque entier naturel n associerait un autre entier $f(n)$, sans pour autant qu'on en ait la formule – comme si ces valeurs étaient fournies par un oracle* quand on en aurait besoin. Même si l'usage de ces valeurs était par ailleurs en tout point conforme à la notion de procédure effective, la seule dépendance envers cet oracle vicierait son caractère. Ce qui est exigé ici, c'est qu'on ait la *formule* de f, autrement dit que cette famille infinie de règles

23. Voir plus haut, § 186.
24. Voir plus haut, § 161.

puisse elle-même être engendrée par *un schéma d'induction*.

§ 194. La notion de procédure effective (2) : exactitude

Restent les conditions (2), (3), (4), et 5a), qui sont toutes liées à la spécification *exacte* (*definite*) de l'espace de problème : les paramètres et les conditions initiales (2), la description de la solution souhaitée (3), et enfin les actions possibles au sein de cet espace où Knuth distingue l'exactitude de la compréhension (4) et celle de l'exécution (5a).

Par ces conditions est posé le double problème de l'ambiguïté et de l'approximation. Knuth donne lui-même l'exemple de recettes de cuisine, qui ne sont pas des algorithmes selon lui, car elles comportent de nombreuses imprécisions. Elles disent par exemple *« ajoutez une pincée de sel »*, ou *« réchauffez le cognac dans une petite casserole »* mais ni la quantité de sel, ni le temps de réchauffement ne sont clairement indiqués. Elles s'adressent à des personnes entraînées à cuisiner, et qui ont une compréhension directe de ces notions. Par ailleurs – pour prolonger Knuth – ce à quoi doit ressembler le plat finalisé est en général décrit de manière très imprécise par des caractéristiques comme « savoureux », « léger », etc. et éventuellement une photo. Or pour Knuth, le test ultime de la bonne définition est qu'« un algorithme doit être spécifié à un degré tel que même un ordinateur puisse suivre ses instructions » – une précision qui ne nous est cependant pas utile puisqu'elle est circulaire vis-à-vis de notre réflexion.

D'abord, il est clair[25] que les procédures résultant d'une investigation rationnelle peuvent être – et sont généralement – du type de la recette de cuisine. Elles s'expriment en général dans des écologies cognitives adaptées à leur finalité, par exemple de mémorisation, de communication, d'enseignement, de vérification et à leur public. Le critère de « bonne définition » est donc relatif à une communauté donnée. Cependant une telle relativité n'est pas incompatible avec l'exigence de Knuth, au contraire.

En effet le problème de la « définition exacte » des significations, dans la perspective de Dewey, n'est rien d'autre que celui de la démarche scientifique tout entière. Celle-ci vise à expliciter et vérifier les inférences contenues dans les systèmes de significations en usage. Le problème de la « pincée de sel » n'existe que si chaque cuisinier l'étalonne d'une manière différente, et si cette différence a une

25. Voir plus haut, section 5.1.

influence sur le résultat final attendu. Si cela n'est pas les cas, il suffit d'observer leur pratique objective pour indiquer à l'ordinateur quelle quantité de sel mesurer.

La condition d'exactitude revient donc, ainsi que nous l'avons déjà remarqué[26], à ce que ces significations et ces règles aient été *prédéfinies avant* l'énoncé de la procédure qui les utilise. Il est interdit de mobiliser arbitrairement de nouvelles significations au cours de cet énoncé, car cela exigerait de l'interrompre afin de détailler les procédures qui permettent de les reconnaître de manière univoque, et ce processus pourrait être reconduit à l'infini – ce qui mettrait en défaut les conditions de finitude vues plus haut. En fait, à proprement parler, de telles interruptions sont autorisées en programmation modulaire – ainsi on fait appel, dans un programme principal, à des fonctions qu'on définit par ailleurs, et cela sur plusieurs niveaux. Ce qui est donc seulement exigé ici par ce critère d'exactitude, est que ce processus de définition en cascade *ne puisse aller à l'infini*, et qu'on aboutisse toujours, au terme d'un nombre fini de digressions, à un sol ferme et fini de règles et de significations que l'exécutant doit comprendre et savoir exécuter.

Ces règles, qui définissent le système dans lequel les procédures doivent puiser leurs moyens, sont en nombre fini dans le même sens que celui vu au développement précédent. On peut présenter ce système comme constitué de familles infinies d'axiomes, mais celles-ci doivent pouvoir être engendrées par un nombre fini de règles d'induction, qui en constituent le véritable socle. Une procédure ne peut être dite effective que si le système qui la sous-tend a été défini de manière inductive – sinon on ne pourrait jamais être sûr qu'une règle douteuse est valide ou non dans le système considéré.

Il y a des pratiques et des expériences humaines qu'on ne sait décrire exactement, mais qui sont pourtant aisément et univoquement compréhensibles par des humains. Par exemple, comment apprendre à un robot de cuisine qu'il a affaire à du sel, et non à du sucre ? Un cuisinier n'aurait qu'à goûter pour le savoir. La solution élémentaire est de placer le sel dans un compartiment du robot prévu à cet effet. Cette responsabilité est dans ce cas déléguée à l'environnement du programme. Mais si on souhaite tout de même que le robot puisse vérifier qu'il s'agit bien de sel, alors il faut lui donner une procédure pour ce faire (et les moyens associés), comme des tests d'apparence visuelle, de chauffe, de pH ou de flamme. Ce principe illustre bien l'idée de Dewey que tout concept scientifique doit pouvoir être résolu en opérations plus élémentaires, pointant vers d'autres concepts. Dans

26. Voir plus haut, § 160 et § 190.

le cas d'un programme informatique, et dans toute procédure d'ailleurs, l'investigation réalise un choix du niveau de représentation de ses objets et de ses actions élémentaires, qui sont alors *hors* de la juridiction du programme.

C'est dans le cadre de cette discussion sur la notion d'*exactitude* qu'il faut comprendre la demande faite par la définition de Knuth, que l'exécution d'une procédure soit une *séquence* d'opérations, progressant par « étapes ». Or on peut imaginer des prescriptions qui se dérouleraient continument, comme par exemple un athlète du 100m qui établirait, au terme d'une série d'entraînements et de longues réflexions, une stratégie très précise pour sa prochaine course, composée de plusieurs opérations s'enchaînant de manière continue : après l'extension de départ, rester souple et se concentrer sur la respiration pendant la phase d'accélération, se redresser une fois la vitesse acquise et maximiser le soutien par les bras, allonger la foulée vers la fin et tendre le buste en avant, etc. Toute cette « chorégraphie » semble faite d'actions continues qui ne correspondent pas aux étapes discrètes de Knuth.

Cependant, ici, si les mouvements sont eux, bien continus, la procédure établie par le sprinter porte sur *le contrôle* qu'il doit exercer sur eux : « allonger la foulée à tel moment », « maximiser le soutien des bras de tel à tel moment », etc. et ces changements de contrôle sont, quant à eux, bien discrets. Ce point illustre bien la thèse de Dewey concernant l'unité de l'expérience. Ce que nous disons être une *action* ne se définit pas par la somme des mouvements unitaires de notre corps ou de notre esprit, mais par l'*impulsion* de tels mouvements dont la coordination est déjà acquise. Aussi n'y a-t-il pas vraiment d'*actions* élémentaires ; celles-ci sont bien plutôt définies à partir du *contrôle* que l'on prend de nos habitudes et de nos réflexes organiques, et opèrent donc toujours sur un fond de continuité élémentaire. Toute procédure est une procédure de contrôle, puisqu'elle est constituée de significations qui pointent vers des opérations *à mener*, *initiant* donc de manière discontinue des changements dans l'expérience, même si ces derniers apparaissent continûment. La procédure élaborée par le sprinter pourrait cependant être non effective s'il n'était pas possible de déterminer de manière exacte *quand* il doit enclencher les différents gestes de sa course, par exemple s'il doit *sentir* quel est le bon moment pour se redresser, et que cette sensation est différente à chaque course, et qu'elle ne peut être mesurée par des opérations précises.

Cela nous amène aux procédures dépendant de situations émotionnelles ou narratives complexes. Là encore, certaines d'entre elles pourraient être effectives.

Toute signification est une opération qui se définit par ses conséquences possibles, et c'est cela qu'il suffit de décrire, par exemple que *« le noir évoque la tristesse »*, même si l'ordinateur ne connaît pas les expériences auxquelles se réfèrent ces mots. Si on réplique que de telles qualités ont trop de connotations possibles pour être énumérées, et cela notamment parce qu'elles entrent dans un dialogue indéfini avec l'histoire individuelle de chacun et celle de sa communauté, alors cela signifie également qu'elles ne peuvent être non plus contrôlées par la démarche scientifique, et assurées dans leurs conséquences. *De facto*, comme le remarque Dewey, la démarche scientifique a exclu de son champ les qualités secondaires (sensations) et tertiaires (émotions, connotations culturelles et sociales).

Ce qui apparaît donc, au terme de cette discussion concernant les conditions d'exactitude, c'est que la procédure dans laquelle se récapitule ou se généralise une investigation est effective (au sens de l'informatique théorique) *à mesure qu*'elle s'exprime dans un système de significations établi scientifiquement. Il s'agit bien d'une relation de proportionnalité, car de nombreuses procédures sont informatisées malgré la part d'ambiguïté et d'approximation qui demeure en elles, et qui crée, dans certaines configurations, les situations de conflit que nous avons examinées plus haut[27].

Une autre manière d'analyser ce rapport est de revenir sur les deux dimensions du contrôle scientifique des significations : 1) la description de protocoles d'établissement de faits primitifs – ce par quoi les significations sont attestées dans l'expérience, et 2) la mise au jour des relations nécessaires entre significations, c'est-à-dire entre les opérations auxquelles elles se réfèrent. Ce deuxième travail est proprement mathématique, puisque les moyens matériels de celles-ci dénotent de pures opérations, libérées de toute contrainte référentielle[28].

Par contre, les opérations d'attestation doivent ultimement se référer à l'expérience, ce qui ne peut être décrit avec exactitude, sinon par davantage de prescriptions opérationnelles, comme on le voit avec les exemples de la pincée de sel. L'expérience vécue elle-même, le « fait brut » dont nous avons vu plus haut le caractère problématique, échappe par définition aux significations, qui ne peuvent que s'y référer. Cela veut dire que la procédure ou le programme informatique ne peut qu'indiquer les opérations à faire pour attester ses objets dans l'expérience, mais ne peut jamais faire *éprouver* ces opérations elles-mêmes. On peut bien in-

27. Voir plus haut, section 5.2.
28. Voir plus haut, § 176 et § 190.

diquer à un robot de cuisine comment reconnaître du sel grâce à un test de pH, mais il dépendra à son tour du bon fonctionnement de ce test. En d'autres termes, les moyens matériels d'un système de significations ne peuvent être absolument exacts, selon l'injonction de Knuth, *que pour autant qu'ils dénotent des opérations pures, c'est-à-dire qu'ils sont partie d'un système mathématique*. L'exactitude est une notion entièrement procédurale, qui ne peut être définie relativement à des significations matérielles[29].

On arrive donc à l'idée que les procédures construites par l'investigation sont programmables *dans la mesure où elle sont mathématiques*, c'est-à-dire en ce qu'on peut décrire leur exécution comme des enchaînements exacts d'opérations, dont on laisse de côté la part référentielle. En y ajoutant l'idée d'une validité exhaustive de la procédure sur son espace de paramètres, qui apparaît en filigrane dans la condition de finitude, nous retrouvons l'idée d'une *science (mathématique) des opérations* dont nous avons déjà esquissé les contours[30].

§ 195. L'informatique comme discipline mathématique

Ce résultat nous ramène à la thèse que l'informatique serait une discipline mathématique, qui fut au centre du débat concernant les méthodes formelles dans les années 1970, déjà évoqué[31]. Cette thèse dont Dijkstra, Hoare et Knuth, sont les principaux partisans, a été récemment étudiée par Tedre (2014), Turner (2018), Primiero (2020). Il s'agit d'une thèse *normative*, c'est-à-dire qu'elle décrit ce que *devrait être* la bonne programmation, et non pas ce qu'elle est dans les faits. Hoare commence son article apologétique sur ce sujet, *Mathematics of Programming*, en définissant les ordinateurs comme des « machines mathématiques », les programmes comme des « expressions mathématiques » et les langages de programmation comme des « théories mathématiques ». Cependant il enchaîne immédiatement :

> Il s'agit là de principes philosophiques et moraux généraux, mais toutes les preuves sont contre eux. Rien n'est comme je l'ai dit, ni ordinateurs, ni programmes, ni langages de programmation, ni même programmeurs. Je trouve les ordinateurs numériques actuels très compliqués et assez mal définis[32].

29. Voir plus haut, § 190.
30. Voir plus haut, section 9.4.
31. Voir plus haut, § 47.
32. (Hoare 1986, p. 116).

Le reste de l'article est consacré à montrer qu'une refondation de la discipline est nécessaire en partant d'une approche mathématique.

Une manière d'interpréter ce caractère mathématique de la programmation consiste, ainsi que le fait Knuth (1974), à concevoir l'informatique théorique comme une branche particulière des mathématiques, consacrée à l'étude des algorithmes. De la même manière qu'il y a en mathématiques une théorie des groupes, une théorie des nœuds, une théorie des fractales, une théorie des probabilités, etc. il y aurait ainsi une théorie des algorithmes, considérés comme des objets formels. On peut en trouver une présentation axiomatique par exemple dans (Sipser 1997).

Cependant une telle interprétation de l'informatique comme mathématique régionale risque de ne pas rendre compte du caractère transverse qu'elle entretient avec les mathématiques qui, tout entières, sont des systèmes d'opérations pures[33]. On peut aller jusqu'à les considérer comme des langages de programmation, ainsi que le fait Martin-Löf :

> C'est la thèse des intuitionnistes (ou constructivistes, j'utiliserai ces termes de manière synonyme) que les notions mathématiques de base, surtout la notion de fonction, doivent être interprétées de telle manière que le clivage auquel nous assistons actuellement entre les mathématiques, c'est-à-dire les mathématiques classiques, et la programmation, disparaît. […]
>
> La différence entre les mathématiques constructives et la programmation ne concerne pas les notions primitives de l'une ou de l'autre, car elles sont essentiellement les mêmes, mais réside dans l'insistance du programmeur pour que ses programmes soient écrits dans une notation formelle afin qu'ils puissent être lus et exécutés par une machine, alors que, dans les mathématiques constructives telles qu'elles sont pratiquées par Bishop (1967), par exemple, les procédures de calcul (programmes) sont normalement laissées implicites dans les preuves.[…]
>
> Ce que je viens de dire sur le lien étroit entre mathématiques constructives et programmation explique pourquoi la théorie des types intuitionniste (Martin-Löf, 1975), que j'ai commencé à développer uniquement avec le motif philosophique de clarifier la syntaxe et la sémantique des mathématiques intuitionnistes, peut tout aussi bien être considérée comme un langage de programmation[34].

Knuth affirme également que l'ouvrage de Bishop, *Constructive Mathematics*, « est de nature entièrement algorithmique[35] », et il pense pouvoir le réécrire tout entier dans un langage de programmation.

33. Voir plus haut, § 176.
34. (Martin-Löf 1982, p. 156).
35. (Knuth 1985, p. 177).

Cependant, si tel est le cas, comment se fait-il qu'on distingue tout de même informatique et mathématiques ? Peut-on se contenter du simple critère du *niveau d'explicitation des preuves* proposé par Martin-Löf ?

Dans un article postérieur de quelques années à celui cité plus haut, consacré également à ces questions, Knuth réinterprète sa définition de l'informatique pour la rendre plus englobante :

> Depuis de nombreuses années, je suis convaincu que l'informatique est avant tout l'étude des algorithmes. Mes collègues ne sont pas tous d'accord avec moi, mais il s'avère que la source de notre désaccord est simplement que ma définition des algorithmes est beaucoup plus large que la leur : j'ai tendance à voir les algorithmes comme englobant l'ensemble des concepts qui traitent des processus bien définis, y compris la structure des données sur lesquelles ils agissent ainsi que la structure de la séquence des opérations exécutées[36].

Une double extension est opérée ici. D'abord, l'informatique n'étudie plus seulement les algorithmes, mais tous les concepts qui s'y rattachent, et notamment les structures de données sur lesquels ceux-ci opèrent. Par ailleurs, les algorithmes (au sens étroit) sont à présent des *processus bien définis*, ce qui suggère que l'exécution même de l'algorithme est un fait objectif qu'on peut inspecter.

Ce léger déplacement focal de la notion de procédure vers celle de processus est éclairé par une remarque complémentaire de Knuth concernant deux spécificités uniques à la réflexion informatique vis-à-vis des mathématiques. D'une part, tandis que les mathématiciens se satisfont de l'effectivité des procédures (de leur parfaite explicitation), les informaticiens se préoccupent également de leur efficacité, c'est-à-dire de leur coût computationnel. Cette idée introduit l'idée de mesure, et donc renforce l'idée que ces processus sont des phénomènes objectifs. D'autre part, la notion d'*état* d'un processus est centrale en informatique, alors qu'elle est absente en mathématique :

> L'autre concept manquant [aux mathématiques] est lié à « l'opération d'affectation », qui modifie les valeurs des quantités. Plus précisément, je dirais que le concept manquant est la notion dynamique de l'*état d'un processus* : « Comment en suis-je arrivé là ? Qu'est-ce qui est vrai maintenant ? Que devrait-il se passer ensuite si je veux arriver au bout ? » Qu'il y ait des états changeants, ou des états instantanés d'un calcul, semble être intimement lié aux algorithmes et à la pensée algorithmique. De nombreux concepts de structures de données, qui sont si fondamentaux en informatique, dépendent très fortement d'une capacité à raisonner sur la notion d'états de

36. (ibid., p. 170).

> processus, et nous nous appuyons également sur cette notion lorsque nous étudions l'interaction de processus qui agissent simultanément[37]. *(Nous soulignons)*

Cette remarque doit être mise en rapport avec la citation précédente : on y retrouve la référence aux structures de données, et aux structure de processus comme partie intégrante du champ de l'informatique. La notion d'état dynamique apparaît donc liée intimement à celle de processus. Il est intéressant pour notre réflexion que Knuth n'éprouve pas le besoin de faire référence à un système matériel, et *a fortiori* à une machine, pour évoquer cette notion. Quand il veut illustrer ce qu'il entend par là, il a recours à la figure du soliloque, suggérant ainsi que son modèle n'est pas l'état d'un système matériel, mais celui d'un investigateur qui a effectué une opération, qui observe le résultat obtenu, et qui s'interroge sur la suite de l'action. Cette description conviendrait parfaitement à Dewey qui insiste sans cesse sur le caractère itératif et progressif de l'investigation[38]. En d'autres termes, l'état fonctionne vis-à-vis d'un processus comme un *interrupteur*, qui observe son déroulement *de l'extérieur*, son impact sur la situation, et décide de le prolonger ou de le modifier.

Cet usage de la notion d'état permet en retour d'éclairer celle de processus. Le processus est bien *l'objet visé* par la reconstruction de l'investigation, *en tant qu'il* est effectué et observé dans son effectuation. Il s'agit par exemple, de la vérification effective d'une preuve mathématique, de la récitation intérieure d'un discours soigneusement préparé, de l'exécution d'un calcul par un élève, etc.

Ainsi, la spécificité de la programmation au sein des mathématiques serait qu'elle *met en action* les moyens procéduraux établis par les théories mathématiques en se préoccupant de leur exécution concrète dans une situation problématique donnée. Gorn (1963) et Caracciolo (1966) poursuivent la même idée lorsqu'ils affirment que la relation de la programmation aux mathématiques serait similaire à celle existant, en linguistique, entre l'étude pragmatique du langage, son étude syntaxique (en mathématiques pures, l'établissement de systèmes axiomatiques) et son étude sémantique (les applications des mathématiques à des modèles)[39]. L'attitude des informaticiens serait en effet caractérisée, par contraste à celles des mathématiciens purs, par leur attention à la *relation* pragmatique entre des utilisateurs et des systèmes symboliques pour la résolution effective de pro-

37. (Knuth 1985, p. 181).
38. *LTI*, p. 41,117.
39. (Gorn 1963, p. 153). Voir aussi (Tedre 2014, p. 38).

blèmes :

> La raison fondamentale de ce lien entre langages de programmation et pragmatique est que les langages de programmation ne sont pas simplement des langages référentiels formels, comme ceux imaginés par les logiciens et autres théoriciens pour définir et référencer tel ou tel type d'entité mathématique abstraite. Ce sont plutôt des langages à utiliser par les programmeurs pour indiquer à un exécutant (*job executor*) quelle tâche (*job*) il doit effectuer. Et c'est essentiellement cette conception de tâche, qui traite d'actions et de processus, qui justifie le mot « pragmatique » en référence à ce domaine de recherche[40].

Nous ne pouvons ici rendre justice à la complexité des réflexions de Gorn sur l'informatique comme pragmatique mathématique ; il parle ailleurs de *pragmatisme cybernétique*[41] : ce qui l'intéresse est que l'informatique a affaire à l'*interprétation* de significations dans un contexte donné, cette interprétation étant elle-même une signification qu'elle peut prendre de manière récursive pour objet. Ces développements complexes mériteraient une étude entière. Nous retenons ici l'idée simplement exprimée par Caracciolo, que l'informatique s'intéresse à *mettre en œuvre* des significations mathématiques en indiquant à un exécutant quelles tâches exécuter dans la résolution d'un problème. L'informatique couvrirait ainsi l'ancien champ des mathématiques pratiques occupées de procédures concrètes de calcul et de mesure[42], par opposition aux mathématiques pures.

§ 196. Le double caractère mathématique de l'informatique

Que tirer de ces diverses indications concernant la distinction entre informatique et mathématiques ?

D'abord, la distinction proposée par Martin-Löf (le niveau d'explicitation supérieur requis par la programmation sur les preuves mathématiques) pointe vers une différence d'usage, au sens de Hersh[43], et non d'essence ; le développement des preuves mathématiques assistées par ordinateur, sur lesquelles nous revenons plus loin dans ce chapitre, les met au même niveau d'explicitation que les programmes informatiques. De la même manière, la recherche d'efficacité proposée par Knuth, n'est ni essentielle, ni exclusive à la programmation relativement aux mathématiques. Certains programmes ne visent pas l'efficacité, mais la correction

40. (Caracciolo 1966, p. 226).
41. (Gorn 1968, p. 340).
42. (Asper 2009). Voir plus haut, § 111.
43. Voir plus haut, § 146.

formelle. Réciproquement, les *Éléments* présentent de nombreuses procédures algorithmiques, et celle de la recherche du plus grand diviseur commun est un modèle d'efficacité computationnelle. Plus fondamentalement encore, on doit objecter à Knuth que les mathématiciens se soucient bien d'efficacité computationnelle quand ils présentent leurs preuves : seulement, celle-ci est relative à l'économie de ressources argumentatives mobilisées, et non au calcul de l'objet mis en évidence par la preuve. Or il n'y a pas de distinction d'essence entre algorithmes de calcul et argumentations, si ce n'est que ces appellations se réfèrent en général à des procédures résolvant des problèmes de niveaux différents, au sens de la théorie de Martin-Löf[44].

Or c'est cette dernière idée qui est essentielle : comme la théorie de la preuve, l'informatique théorique semble être une mathématique de second ordre, en ce qu'elle décrit le procès même de l'investigation et ses propriétés, ou plutôt de l'exécution de la procédure reconstruite après l'investigation. Cela provient du fait que la programmation elle-même est, comme la démonstration, un acte de retour réflexif sur un jugement complexe pour en expliciter tous les moments nécessaires. C'est en cela qu'elle peut apparaître à Gorn et à Caracciolo comme une pragmatique, qui étudie la *relation* des significations aux actes d'un exécutant, ou encore comme l'étude des règles *en tant qu'elles donnent* une séquence d'opérations pour résoudre un problème, pour reprendre les termes de la définition par Knuth d'une procédure effective[45].

C'est donc cet exécutant qui est central en informatique : c'est à lui que s'adressent les prescriptions de la programmation, et c'est son comportement résultant (la ratiocination) qu'étudie l'informatique théorique. Si on met de côté les procédures interactives, cette étude peut sembler avoir pour objet fondamental les transformations de cet exécutant d'un état initial à un état final, qui se représentent en mathématiques par les *fonctions de transition*, opérant sur un espace d'états associé, lui-même défini de manière plus ou moins complexe grâce aux « structures de données » évoquées par Knuth, comme par exemple des listes* ou des dictionnaires*. Du moment qu'on lève la restriction concernant les procédures interactives, l'informatique apparaît bien comme la mathématique de la résolution de problème, c'est-à-dire de transformations de situations opérées par un exécutant en leur sein, en tant qu'il exécute des règles prédéfinies,

44. Voir plus haut, § 147.
45. Voir plus haut, p. 752.

afin d'obtenir une situation cible, ce qui rejoint notre interprétation contractuelle de la programmation[46].

En ce sens, l'informatique théorique est bien une mathématique régionale, ainsi que le veut Knuth, qui définit ses propres concepts en fonction des champs de problèmes qu'elle rencontre. Elle ne manque certes pas d'utiliser des concepts mathématiques déjà disponibles – les moindres n'étant pas ceux de nombre, de fonction, d'ensemble, de graphe, etc. mais elle peut également inventer les siens propres – ainsi la théorie de la concurrence est entièrement informatique. À un niveau pratique, la programmation ne part pas de significations mathématiques qu'elle tente d'appliquer à un problème afin de le résoudre, mais au contraire (nous nous plaçons ici dans le cas simple où le problème serait déjà résolu) elle part des significations qu'elle trouve dans cette résolution afin de les représenter systématiquement, c'est-à-dire comme la composition de moyens procéduraux exacts dans un espace de conditions décrit exhaustivement. Quand le problème est déjà exprimé dans un cadre systématique (par exemple s'il s'agit d'un problème de mathématiques ou d'ingénierie) elle peut directement mobiliser leurs moyens procéduraux.

Dans le cas contraire cependant – par exemple lorsqu'il s'agit de réfléchir au budget d'une nouvelle usine – le programmeur doit elle-même inventer et vérifier un mini-système de significations adéquat pour exprimer cet espace de conditions et ces moyens procéduraux, avant d'y détailler sa solution.

C'est en tous les cas dans ces entreprises, où on cherche à systématiser une situation *non systématique*, que la programmation dévoile son caractère mathématique propre. Il s'agit pour le programmeur, d'une part de négocier la description de la situation et de ses conditions avec le maître d'ouvrage, d'autre part de s'assurer que les moyens procéduraux requis sont à la portée de l'exécutant (la machine). Le langage procédural ainsi conçu peut être considéré comme une « mini-théorie » mathématique, centrée sur la classe de problèmes qu'elle vise à résoudre, et qui permet d'accomplir les visées de second ordre évoquées ci-dessus (preuve de correction, automatisation de la recherche, recherche d'optima de procédures, etc.).

Par ailleurs, rien n'empêche que de tels mini-langages soient mis en commun et factorisés à un certain niveau d'abstraction où leurs structures communes sous-jacentes sont rendues explicites. C'est ainsi, nous l'avons dit, que de nombreux langages comme Modelica ou applications comme les tableurs* peuvent être mobilisés pour de très larges classes de problèmes.

46. Voir plus haut, section 3.3.

Une telle idée est en forte résonance avec l'interprétation des langages de programmation comme des *théories mathématiques d'opérations*, proposée par le philosophe de l'informatique Ray Turner[47], que nous avons déjà sollicité[48]. Les langages sont bien en effet des théories, car ils sont des systèmes de significations contrôlés dans lesquels s'expriment des procédures. Ces significations portent de manière centrale sur des opérations et sur les structures sur lesquelles elles agissent (types, objets, classes, contextes, etc.). C'est la diversité possible des manières de former des concepts d'opérations qui explique celle des langages de programmation. Il faut noter que pour Turner ce caractère mathématique des langages de programmation est indépendant de leur présentation axiomatique et déductive : il y a en effet peu de langages qui ont été complètement vérifiés, et Turner note que leur spécification est souvent alambiquée et complexe. Il n'en reste pas moins que selon lui l'essentiel n'est pas là, car il pense, comme Hersh, que les axiomatisations surviennent tardivement dans les disciplines mathématiques. Pour lui, les langages de programmation sont mathématiques en ce que leurs objets peuvent être explorés à l'aide de notions mathématiques comme celle de classe d'équivalence et par des procédures de preuve[49]. Pour nous, cela tient à leur caractère d'opérations *pures*, mais Turner bien sûr ne se place pas dans le même cadre que nous.

En synthèse de cette discussion, nous pouvons avancer que la relation de l'informatique aux mathématiques est double.

D'une part, la programmation développe ses moyens procéduraux propres pour résoudre les problèmes qui lui sont assignés lorsqu'elle ne les trouve pas disponibles dans les théories mathématiques, comme le montre l'exemple de Catala. Elle développe donc des théories mathématiques *ad hoc*, qu'on pourrait qualifier d'*appliquées*, car elle se trouve alors en analogie certaine avec les pratiques d'invention procédurale qu'on trouve dans les sciences et les techniques. Il s'agit en général de théories conceptuellement modestes et construites de manière pragmatique – ce qui explique la difficulté qu'il y a à prouver leur cohérence et leur non ambiguïté dès qu'elles sont assez riches.

D'autre part, elle est une mathématique de second ordre et, comme la théorie de la preuve, elle s'intéresse aux *actes mathématiques*. C'est en cela qu'elle peut apparaître à Gorn et Caracciolo comme une pragmatique, qui relie les significations

47. (Turner 2010, p. 74, 76, 78).
48. Voir plus haut, § 46.
49. (Turner 2010, p. 74).

aux actes ; cependant l'analogie linguistique nous semble trompeuse, car la science du langage n'étudie pas en général des significations qui sont *déjà procédurales*. Quoiqu'il en soit, ce qui est crucial ici est la référence à un exécutant d'une tâche qui transforme son propre état et, potentiellement, celle de la situation elle-même.

La section suivante vise à développer plus concrètement ces idées par l'esquisse d'une épistémologie positive de l'informatique théorique et de l'informatique pratique, où nous entendons par là l'interprétation de leurs grands domaines de recherche telles que nous pouvons les observer aujourd'hui.

11.2 Esquisse d'une épistémologie de l'informatique

§ 197. L'informatique théorique et ses enjeux pratiques

Sur la base de cette interprétation de la nature mathématique de la programmation, il est possible de porter un nouveau regard sur les savoirs positifs que comprend l'informatique, en prenant pour fil conducteur le lieu où ces savoirs s'organisent, c'est-à-dire l'informatique théorique.

Celle-ci s'est structurée historiquement en deux disciplines, l'étude des méthodes formelles d'une part, et celle de la complexité des algorithmes d'autre part. Elles commandent, par exemple, la division d'un ouvrage de référence tel que le *Handbook of Theoretical Computer Science* en deux volumes distincts, avec la justification suivante :

> Il existe plusieurs rôles différents pour la « théorie » en informatique. D'une part, elle fournit les fondements mathématiques nécessaires à l'étude des systèmes et des algorithmes formels utiles. D'autre part, elle fournit des concepts et des langages permettant de capturer, en termes algorithmiques et descriptifs, l'essence de tout système depuis la spécification jusqu'à la mise en œuvre efficace. [...] La version actuelle du manuel est présentée en deux volumes :
>
> Vol. A : algorithmes et complexité
>
> Vol. B : modèles formels et sémantique
>
> Elle reflète plus ou moins la division observée en informatique théorique entre les recherches orientées vers les algorithmes et celles orientées vers les descriptions, et qu'il semblait naturel de suivre dans cet ouvrage[50].

On peut noter à cet égard que le domaine des méthodes formelles est de tradition plutôt européenne, tandis que celui de la complexité algorithmique est plus

50. (Van Leeuwen 1990, vol. A, p. i.)

développé aux États-Unis[51].

Chacun de ces deux domaines représente un aspect du traitement mathématique de la résolution procédurale de problèmes. Cela est assez évident pour le domaine algorithmique, ainsi que l'exprime par exemple, le département d'informatique théorique du MIT dans sa propre présentation :

> Ce domaine comprend deux sous-domaines : la théorie des algorithmes, qui comprend la conception et l'analyse des procédures de calcul ; et la théorie de la complexité, qui porte sur les recherches pour prouver qu'aucun algorithme efficace n'existe dans certains cas, ainsi que sur le système de classification des tâches de calcul. Le temps, la mémoire, l'aléatoire et le parallélisme sont des mesures typiques de l'effort de calcul. L'informatique théorique est un pont naturel entre les mathématiques et l'informatique[52].

Quant au domaine des méthodes formelles, il est bien décrit par cette remarque de van Leeuwen :

> Les cadres mathématiques qui se sont révélés inestimables pour l'informatique sont de plus en plus utilisés par de nombreuses autres disciplines. Partout où les notions d'information ou de traitement de l'information sont identifiées, des questions de représentation et de calcul peuvent être formalisées en termes informatiques[53].

Cette dualité disciplinaire de l'informatique théorique nous semble relever d'une interprétation plus générale dans le cadre que nous avons construit. Ce qu'étudient en effet les théoriciens de l'informatique s'ancre dans les problèmes pratiques auxquels sont confrontés les programmeurs.

L'algorithmique et l'étude de la complexité visent à répondre au besoin pratique évident de *trouver les bonnes procédures ou méthodes de résolution de problèmes*, ce qu'on appelle l'*heuristique*. Dans le cas des méthodes formelles, l'interprétation est moins directe. Celles-ci visent à vérifier exhaustivement un programme, c'est-à-dire à en apporter une preuve de correction. Cela ne se fait pas par une validation expérimentale, mais par sa confrontation à sa spécification, c'est-à-dire à une autre signification. Il s'agit donc d'un examen purement logique du concept en question et de son comportement opérationnel. Les méthodes formelles répondent au besoin pratique des programmeurs de *trouver les bons concepts par lesquels décrire les problèmes et les situations* de manière entièrement procédurale,

51. (Dean 2016).
52. Présentation en ligne du département d'informatique théorique du MIT, consulté le 20 septembre 2023.
53. Van Leeuwen (1990, vol. A, p. i).

en termes d'opérations pures.

On a ainsi la correspondance suivante :

Théorie	Pratique
Algorithmique	Heuristique
Méthodes formelles	Conception (design)

La première colonne reprend les deux disciplines majeures de l'informatique théorique, et la seconde, les enjeux pratiques des informaticiens : trouver de bonnes méthodes et de bonnes représentations pour programmer. Mais c'est horizontalement qu'il est plus important de lire ce tableau. L'algorithmique et l'heuristique s'occupent tous deux d'étudier les problèmes eux-mêmes et leurs « bonnes » solutions. Concevoir des algorithmes est une forme essentielle et pure de programmation, où les phases de spécification et de codage sont réduites à leur strict minimum afin que l'on puisse se concentrer sur l'aspect purement procédural (mathématique) de la programmation ; on ne s'y occupe que de trouver les meilleurs méthodes pour résoudre des problèmes canoniques et établir une théorie de ces problèmes, c'est-à-dire à établir leurs interdépendances, comme par exemple que le problème du cycle hamiltonien* peut être réduit en temps polynomial au problème de satisfiabilité booléenne (SAT)* et à consigner leurs meilleures solutions, qu'elles soient mesurées en terme de temps et de mémoire (les critères les plus usuels), ou encore d'intelligibilité, de sécurité, etc. Quant aux méthodes formelles et à la conception (ou *design*), elles ont trait à la représentation et à la vérification de systèmes de moyens procéduraux et d'espaces de conditions qui peuvent être parcourus de manière inductive.

On peut remarquer que, dans la logique de Dewey, bien qu'on ait affaire ici à des moyens procéduraux de part en part, ces deux groupes de disciplines correspondent à la dualité entre moyens procéduraux et moyens matériels. L'algorithmique et l'heuristique s'occupent de concevoir les procédures (les « techniques ») tandis que les méthodes formelles et le design s'occupent de concevoir les significations qu'elles manipulent (les « outils »), et qui permettent de décrire la situation problématique – de reconnaître le problème procédural traité dans l'expérience.

Les quatre développements suivants développent ces idées et les rendent plus concrètes, en passant en revue les quatre domaines cités : algorithmique (§ 198), heuristique (§ 199), méthodes formelles (§ 200), et design (§ 201). Ils ont pour objectif d'esquisser sur ces bases une épistémologie de l'informatique. Cette esquisse répond partiellement à notre question directrice concernant la programma-

tion comme forme de connaissance. Mais elle vise aussi à montrer la place de la programmation et des mathématiques constructives dans l'économie générale de la connaissance.

Il ne peut s'agir ici d'être systématique, par manque d'espace et de temps ; ces domaines sont si vastes et se réexpriment si rapidement eux-mêmes au gré des progrès conceptuels réalisés par les informaticiens, qu'il ne peut être question ici de certitude, mais seulement de suggestions.

§ 198. Les concepts fondamentaux de l'algorithmique

Les deux développements suivants, consacrés aux notions d'algorithme et d'heuristique vont s'appuyer sur deux ouvrages d'algorithmique. Le premier, d'Aho, Ullman et Hopcroft ([1974] 2006), est un classique qui passe en revue les principaux algorithmes utilisés en informatique – tris, recherche de chemins, multiplication de matrices, algorithmes numériques, etc. Le second, de Michalewicz et Fogel ([2000] 2004), est plus original : il se propose, non seulement de passer en revue les algorithmes importants – qui incluent, du fait des progrès réalisés, également les réseaux de neurones, les algorithmes évolutifs et les systèmes multi-agents – mais également de prodiguer des conseils et méthodes pour mieux résoudre les problèmes.

Le chapitre 2 du livre d'Aho et Ullman se présente comme une introduction aux « structures de données élémentaires et aux techniques de programmation souvent utilisées dans les algorithmes efficaces[54]. » Ses quatre premières sections traitent en effet des structures de données : listes*, queues*, piles*, ensembles, graphes et arbres. Les quatre autres portent les titres suivants : récurrence – diviser-pour-régner (*divide-and-conquer*) – équilibrage de charge (*load-balancing*) – programmation dynamique. Nous nous proposons de tenter de comprendre la raison de ces choix parmi tous les concepts disponibles de l'informatique.

Une procédure n'est en général intéressante à consigner que si elle présente un gain d'efficacité important par rapport à d'autres options possibles, ou tout simplement par rapport à l'investigation initiale, pour un problème important ou fréquemment rencontré.

Cela est le cas dans les tâches répétitives. Une procédure qui trouve le moyen de diminuer le nombre global de répétitions à réaliser, par exemple en modifiant

54. (Aho, Ullman et Hopcroft [1974] 2006, p. iii).

l'ordre interne des tâches, présente un intérêt certain. Même s'il n'y a pas un tel gain, il peut être intéressant de bien mettre en valeur la structure itérative de la procédure, afin d'identifier les tâches unitaires auxquelles elle fait appel, et chercher à les optimiser individuellement. Quelques miettes d'efficacité gagnées sur une tâche unitaire peuvent au global générer un immense gain.

La récursivité[55] est le modèle général des procédures répétitives, car la réitération d'une tâche est une forme de récursivité. Elle ne peut en effet donner un résultat que si elle se termine ; elle doit donc dépendre d'une forme de limite sur une accumulation sur laquelle elle agit[56] – par exemple vider un sac de billes une à une agit sur le nombre de billes dans le sac, et monter un escalier marche à marche agit sur le nombre de marches restant à gravir. De nombreuses procédures qui n'ont pas l'air répétitives ont ainsi une structure récursive. Par exemple, pour décrire un chemin à travers un labyrinthe ou une grande ville, il suffit d'en consigner les directions successives dans une liste (par exemple : `[D,A,A,G,A,D,...]` où `D,A,G` signifient respectivement *droite, en avant, gauche*). L'algorithme consiste alors à parcourir cette liste de manière récursive jusqu'à la fin, en exécutant les instructions une à une, dont la diversité est très limitée (avancer, tourner à gauche, à droite, etc.). Il y a valeur à factoriser ainsi l'exécution de ces instructions, car elles pourraient requérir des sous-instructions qu'il serait inefficace de répéter à chaque fois (par exemple : traverser les rues au feu vert seulement). Une telle séparation de la liste des instructions du lieu de leur exécution permet également de simplifier la *maintenance* de la procédure, car de nouvelles consignes concernant les instructions unitaires n'ont plus à être ajoutées qu'à un seul endroit, plutôt qu'à chacune de leurs occurrences.

On voit dans cet exemple la première apparition d'une structure de données, la liste. La liste est le graphe le plus élémentaire qui soit, la ligne orientée. De nombreux algorithmes se présentent comme des récurrences sur des graphes plus compliqués, et notamment des arbres : c'est pour cela que ces trois structures sont mentionnées par Aho et Ullman. On voit à l'exemple du chemin dans le labyrinthe que les algorithmes ont besoin de ces structures non pas tant comme de données ou matériaux sur lesquelles agir, mais comme de *memoranda* de la suite diverse et contingente des instructions à répéter, qu'il s'agit de mettre de côté afin de concentrer l'algorithme sur ses structures récursives et ses opérations fonda-

55. Voir plus haut, § 147.
56. Voir plus haut, § 162.

mentales. Cette remarque permet de se rappeler qu'il n'y a pas, chez Dewey, de distinction fondamentale entre *donnée* et *instruction*. Toute donnée est un signe d'opérations possibles, comme par exemple le nombre 3 indique trois répétitions d'une opération.

Il y a cependant des procédures importantes qui ne présentent pas de manière évidente une structure répétitive. Les preuves mathématiques en sont un exemple, mais on trouve également de tels cas de figure dans chaque domaine technique et scientifique. Parfois les choses ne sont pas du simple ordre de la routine, et c'est justement la difficulté et l'originalité de leur cheminement qui invite à les consigner. Cependant même les preuves cherchent à se simplifier et à se factoriser, d'une certaine manière, en s'appuyant sur des résultats antérieurs. Leur structure est ainsi, dès que possible, modulaire. Cette modularité ne dépend pas d'elle, mais de l'utilité plurielle qui a déjà été reconnue à certaines de leurs étapes. La réorganisation théorique d'un champ, on l'a vu[57], consiste ainsi souvent à réorganiser ses résultats sous la forme d'un progrès cumulatif, ainsi que les *Éléments* d'Euclide en donnent le modèle.

Cette approche synthétique correspond à la programmation dynamique dans le chapitre étudié. Elle se différencie des récurrences analytiques qui décomposent le résultat recherché en unités plus petites jusqu'à arriver à des problèmes élémentaires. Pour certains problèmes, cette démarche est inefficace, car au cours de l'exploration du graphe, les mêmes problèmes intermédiaires seront rencontrés de multiples fois. Une approche synthétique est alors préférable, qui procède « de bas en haut », en commençant d'abord par la résolution systématique des sous-problèmes élémentaires, et en remontant progressivement vers les problèmes composés. C'est la stratégie que nous avons utilisée lors de l'optimisation de notre problème de change[58].

Aho et Ullman disent que la programmation dynamique est *tabulaire*, car elle exige de mémoriser des résultats intermédiaires dans une table[59]. Une table peut être modélisée comme une pile ou une queue, c'est-à-dire comme une structure de données dynamique, que l'algorithme ne se contente pas de parcourir en lecture, mais qu'il peut également modifier. Les piles et les queues sont les structures dynamiques les plus simples (une queue est suffisante pour simuler n'importe quelle

57. Voir plus haut, § 182.
58. Voir plus haut, § 28.
59. (Aho, Ullman et Hopcroft [1974] 2006, p. 67).

machine de Turing), et permettent ainsi aux algorithmes d'avoir une mémoire.

Ainsi donc, les notions élémentaires d'algorithmique proposées par Aho, Ullman et Hopcroft nous semblent pouvoir être justifiées à partir de l'idée même de procédure. Il y a en effet plusieurs lieux où il est possible d'optimiser un processus, selon qu'il a une structure répétitive ou simplement modulaire ; selon qu'il vaut mieux procéder analytiquement ou synthétiquement ; selon le type de graphe qui lui est le mieux adapté ; selon que ce graphe est dynamique ou non. Enfin, les deux sujets restants : *diviser pour régner* et l'équilibrage de charge présentent un seul précepte générique valant pour toute conception d'algorithme, qui est de rechercher des décompositions équilibrées en masse, qu'il s'agisse de récurrences, de modules, de structures de données, car c'est cet équilibrage qui maximise l'efficacité globale.

Ces concepts permettent de pointer vers les raisons de l'universalité de la programmation, c'est-à-dire sa capacité à mettre en procédure la part mathématique et opératoire du résultat de toute investigation ; ils dérivent directement des principes de la science des opérations[60].

§ 199. Les heuristiques

L'ouvrage de Michalewicz et Fogel ([2000] 2004) ne revient pas sur ces concepts fondamentaux, mais vise plutôt à éduquer le lecteur sur des stratégies plus globales de résolutions de problèmes. Le titre *How to Solve it : Modern Heuristics* est une référence appuyée à l'ouvrage du même nom de Pólya ([1945] 1990). George Pólya, que nous avons déjà rencontré[61], est un mathématicien hongrois qui, dans cet ouvrage de didactique et d'autres subséquents, remet à l'honneur le terme peu connu d'*heuristique*[62] en esquissant une méthode générale et en donnant de nombreux conseils pour la résolution de problèmes mathématiques. Cet ouvrage devient une référence dans les cercles de la psychologie cognitive et de l'intelligence artificielle qui se développent dans les années 1950. Il est notamment repris par Newell (qui eut Pólya pour professeur) et Simon dans leurs recherches.

C'est cette dimension réflexive développée par Michalewicz et Fogel ([2000] 2004) qui nous intéresse ici. Les auteurs identifient quatre dimensions qui rendent

60. Voir plus haut, § 162.
61. Voir plus haut, § 146.
62. Sur l'histoire du terme, voir (Schepers 1971).

les problèmes difficiles à résoudre :

1. L'espace de problème est trop grand pour être exploré systématiquement.
2. Le nombre de contraintes à satisfaire est élevé, et ces contraintes sont hétérogènes, certaines par exemple étant discrètes (« *tous les chemins doivent passer par Paris* »), et d'autres des recherches d'optimisation (« *le chemin doit être le plus court possible* »).
3. Le problème est difficile à représenter exactement, et tout modèle introduit des simplifications et des approximations dont il faut tenir compte.
4. Le problème varie dans le temps (au cours de sa résolution) ou comprend une part d'incertitude.

Ces difficultés justifient qu'on parle ici d'heuristique. On trouve deux définitions du terme chez les cognitivistes qui, si elles s'accordent souvent, sont néanmoins distinctes. Dans les deux cas, le terme est presque toujours employé au pluriel, reconnaissant implicitement par là que la collection d'objets désignés ainsi ne saurait devenir une science ni même une technique à part entière.

On peut d'abord définir les heuristiques comme des méthodes, des règles, ou des conseils pour trouver les bonnes procédures. Il s'agit, en quelque sorte, de méthodes à trouver les méthodes. Il leur est caractéristique de procéder *interactivement* avec le problème posé, en ce sens que ce problème est-lui même « mal défini », puisqu'il ne permet pas d'être résolu tel quel. Il s'agit donc, dans une démarche heuristique, d'obtenir plus d'information sur la situation, typiquement en posant des questions, de préciser l'objectif, de trouver des compromis entre plusieurs objectifs et contraintes[63]. Il faut également choisir quelles tâches essayer prioritairement, mais également « surveiller » ces tâches, afin d'être capable de changer de cap si elles n'aboutissent pas, ou si la situation évolue. Ce sens est très présent chez Simon[64] puisque, pour lui, les procédures de résolution de problèmes sont « des moyens procéduraux, des processus pour trouver des programmes qui atteindront les fins visées[65] ». L'heuristique ici, est la méthode de l'investigation elle-même.

Dans un autre sens, les heuristiques sont des procédures complémentaires aux algorithmes – des programmes qui, contrairement aux algorithmes, ne sont pas assurés de donner une solution en un temps garanti, ou une solution exacte, mais

63. (Dörner 1983, p. 93-100).
64. Voir plus haut, section 4.2.
65. (Simon [1967] 1977, p. 150).

dont l'expérience a montré qu'ils pouvaient donner, *la plupart du temps*, une solution *acceptable* en un temps *raisonnable*. Ce sens est aussi très présent chez Simon, qui parle alors de « méthodes faibles ». Tous les algorithmes d'apprentissage sont de cette nature, puisqu'ils se présentent comme des recherches du minimum d'une fonction. Dès que celle-ci est suffisamment complexe, il n'est en général pas possible de garantir qu'on ait trouvé son minimum absolu, mais seulement un minimum local.

Ces deux sens d'*heuristique* s'articulent l'un à l'autre en ce que les « méthodes pour trouver des méthodes » sont des méthodes faibles. S'il s'agissait de méthodes fortes, on pourrait parler, figurativement, de « compilateurs », c'est-à-dire de méthodes qui, à partir d'une spécification initiale, construisent rigoureusement une méthode certaine d'aboutir. Réciproquement, si on ne dispose que d'une bibliothèque de méthodes faibles pour résoudre un problème, la réflexion stratégique est d'autant plus nécessaire pour choisir laquelle appliquer et surtout l'adapter à la situation afin d'espérer obtenir une solution acceptable. C'est le *motto* exprimé par les auteurs, après l'évocation d'un résultat mathématique prouvant l'incapacité d'une quelconque représentation à optimiser l'exploration du problème du voyageur de commerce*, un problème majeur en algorithmique :

> Plutôt que […] doucher notre enthousiasme à trouver de bons algorithmes […], regardons les choses avec plus d'optimisme. Premièrement, nous savons que nous devrons utiliser toutes nos connaissances sur le problème afin de concevoir un algorithme plus efficace qu'un échantillonnage aléatoire aveugle. Ainsi, il est de notre responsabilité de chercher à savoir tout ce qu'il est possible sur le problème et d'essayer d'incorporer ces connaissances dans des procédures utiles. Deuxièmement, nous savons qu'il n'y aura pas une unique meilleure façon de résoudre le problème. Au contraire, il peut y en avoir plusieurs, et encore plus de très bonnes ! Quel que soit le problème auquel nous pouvons être confrontés, il n'y a pas à se préoccuper de lui appliquer de force une représentation ou un opérateur de recherche particuliers. Notre quête est de concevoir le bon outil qui correspond au bon problème[66].

Ainsi, Michalewicz et Fogel alternent les chapitres consacrés à des techniques algorithmiques proprement dites et ceux qui développent des thématiques informelles concernant l'investigation, dans l'esprit des conseils heuristiques de Pólya, voire des *Règles pour la direction de l'esprit* de Descartes. C'est pour cette raison qu'ils mettent beaucoup en avant les algorithmes génétiques ou évolutifs, car selon eux ils sont des sortes de « couteaux suisses » capables de s'adapter à des gammes

66. (Michalewicz et Fogel [2000] 2004, p. 162).

très variées de problèmes par leur grande paramétrabilité[67]. Il s'agit d'une famille d'algorithmes qui, selon ces auteurs, laissent de la place à *l'intelligence* et qu'ils opposent aux techniques ultra-efficaces mais ultra-spécialisées que la recherche algorithmique a nécessairement favorisées en plus de cinquante années de travaux.

Ce parti-pris de Michalewicz et Fogel nous oblige à laisser inachevée cette courte esquisse, car elle nous entraînerait dans des développements trop longs. En effet, leur ouvrage est écrit au début des années 2000, et si un chapitre est consacré aux réseaux de neurones, il est loin d'anticiper leurs succès, qui donnent aujourd'hui l'impression d'une « monoculture » qui gagne du terrain quel que soit le domaine rencontré, grâce entre autres à la puissance brute numérique et aux données massives qui leurs sont fournies. Nous nous basons ici sur l'article suggestif de Cardon, Cointet et Mazières (2018), qui mériterait une analyse plus poussée. Comment interpréter cette annulation apparente du besoin d'investigation par des techniques procédurales uniformes ? Ainsi, par exemple, tous les cadres d'analyse de la vision humaine ou du langage qui avaient été développés pendant trois décennies pour tenter de les mettre en procédure se sont trouvés balayés par l'effectivité de l'apprentissage profond. Cardon, Cointet et Mazières évoquent le « sentiment déstabilisant de voir *une technique sans théorie* se substituer aux efforts de modélisation patiemment conduits depuis des années[68] », qui semble tout fait faire écho à l'annonce de la « fin de la programmation[69] ». Nous revenons brièvement en conclusion sur la question difficile de la possible substitution de l'investigation par des méthodes d'apprentissage statistiques.

§ 200. Les méthodes formelles

Les méthodes formelles[70] ont pour finalité de vérifier la correction des programmes informatiques, en fonction d'une spécification qu'elles doivent permettre d'exprimer également. Comme on l'a vu, certaines permettent également de guider le développement du programme à partir de sa spécification, voire de l'en *dériver* directement, mais il s'agit là d'un prolongement de leurs finalités premières, de

67. (Michalewicz et Fogel [2000] 2004, p. 163). Sur les algorithmes génétiques, voir leurs développements à partir du chapitre 6.
68. Cardon, Cointet et Mazières (2018, p. 207) *(Nous soulignons)*.
69. Voir plus haut, § 25.
70. Voir plus haut, section 3.1. Pour une introduction historique et conceptuelle sur ce domaine, voir (Astarte 2019 ; Bjørner et Havelund 2014 ; C. B. Jones et Astarte 2017 ; MacKenzie 2004).

spécification et de vérification.

On a affaire donc ici à une démarche scientifique dans la définition exacte qu'en donne Dewey[71] : un programme étant une signification procédurale, il s'agit de le vérifier rigoureusement avant de le mettre à disposition d'utilisateurs dans un système informatique. La recherche en méthodes formelles vise à définir les approches qui permettent d'établir rigoureusement les significations élémentaires d'un système informatique. Il n'est pas étonnant, de ce fait, qu'elles se soient d'abord développées dans le domaine de la conception de langages de programmation et de matériels informatiques (processeurs, systèmes embarqués), puisqu'il s'agit là des systèmes élémentaires dans lesquels on exprime des procédures[72]. Elles peuvent être cependant appliquées à n'importe quel programme. Comme nous l'avons déjà mentionné, le logiciel de commande de la ligne de métro 14 à Paris, entièrement automatisée, a été développé formellement[73].

Par *spécification formelle*, on entend que les propriétés attendues du fonctionnement du programme doivent être exprimées dans un système de significations mathématique[74]. Il est en effet aisé de donner une spécification informelle du comportement attendu d'une procédure, en en donnant des exemples, ou en le décrivant en langage naturel. Les exemples ne permettent cependant pas de couvrir systématiquement tous les cas de figure possibles, et le langage naturel, on l'a vu, est intrinsèquement ambigu. En exigeant de décrire les attendus du programme mathématiquement, on demande donc d'établir une correspondance entre la procédure, telle qu'elle est décrite dans son système de significations d'origine (« le langage de programmation »), et un autre système de spécification, déjà bien connu, comme la logique du premier ordre ou la théorie des ensembles, dans lequel elle est susceptible d'une description systématique. C'est pour cela qu'on parle de *sémantiques formelles*, car il s'agit pour ces méthodes d'établir une telle correspondance – une expression qui a pour avantage de montrer combien les objets informatiques sont avant tout des significations.

Jones et Astarte distinguent deux grands usages des méthodes formelles : soit il s'agit d'établir la correction unitaire d'une procédure donnée, soit il s'agit de vérifier la correction globale d'un système de procédures qui doivent pouvoir être

71. Voir plus haut, § 182, et plus particulièrement la citation 701.
72. (Astarte 2019, p. 2-8).
73. (Abrial 2013).
74. Voir l'exemple du portillon, plus haut § 55.

composées entre elles, c'est-à-dire d'un langage de programmation[75].

Concernant d'abord le second usage, il faut noter que toutes les procédures du langage proposé doivent être décrites dans le même système de significations sous-jacent, qu'on peut appeler le « langage-machine ». La vérification du langage revient alors à montrer que ces procédures ont bien le comportement attendu quand on les traduit dans ce langage. La vérification porte donc ici sur cette traduction elle-même, c'est-à-dire sur le compilateur du langage – mais il ne faut pas s'y tromper : si des problèmes sont rencontrés, par exemple si la traduction se révèle ambiguë ou si des incohérences sont décelées, c'est souvent que le jeu de procédures considéré n'est pas encore assez bien défini. Le langage modèle dans lequel on décrit le comportement des procédures est celui des inférences élémentaires qu'il est demandé d'accepter et de savoir retranscrire (les « transformations de la machine »). Deux méthodes ont été développées pour ce faire, les sémantiques opérationnelle et dénotationnelle, mais leur distinction ne nous intéresse pas ici.

Le premier usage des méthodes formelles cité plus haut, qui est de vérifier la correction d'un programme particulier, permet de bien mettre en évidence la dimension logique pure de ces travaux de vérification. Dans la sémantique dite axiomatique, qui n'est rien d'autre que la logique de Hoare[76], et qui fut spécifiquement développée pour cet usage, il n'est plus question de machine et de mémoire, sinon implicitement, mais *du problème lui-même* qu'on cherche à résoudre, de son environnement initial et de la situation finale à laquelle on souhaite aboutir, tous deux exprimées par des propositions logiques que l'exécution de la procédure doit satisfaire.

Nous avons déjà mis en avant ce point central lors de notre analyse du problème du portillon[77]. Un autre exemple concerne l'attribution d'une clé magnétique de chambre au nouveau client d'un hôtel. La pré-condition de cet événement est que la chambre soit bien réservée au nom du client, et la post-condition que la clé ouvre la porte de la chambre en question[78]. Le comportement logique de chaque instruction du langage de programmation doit avoir été préalablement défini, de sorte que vérifier que la procédure résout bien le problème revient à dériver formellement la post-condition requise à partir de la pré-condition et de ses règles

75. (C. B. Jones et Astarte 2018, p. 14-15).
76. Voir plus haut, § 138.
77. Voir plus haut, § 55.
78. (D. Jackson 2006, p. 185-203).

d'inférence.

Cette méthode axiomatique s'est prolongée et enrichie pour exprimer des spécifications plus complexes que la simple description d'une situation initiale du problème et d'une situation finale satisfaisante. Des conditions d'invariance permettent de décrire des relations structurelles entre significations auxquelles la procédure doit se conformer durant tout le temps de son exécution. Par ailleurs, une procédure est souvent composée de plusieurs sous-procédures qui doivent également avoir des comportements cohérents. Ainsi, des langages de spécification formelle, comme B, Z, VDM ou Alloy ont été développés qui permettent de décrire les relations logiques qui doivent tenir entre tous les objets du programme, de manière invariante ou dynamiquement. Ces langages, développés initialement pour être des notations de spécifications auxquelles comparer les programmes[79], se sont révélés essentiels pour vérifier la *robustesse interne des spécifications elles-mêmes*[80]. Par exemple la spécification de la gestion des clés de chambre d'un hôtel doit s'assurer qu'aucune intrusion non autorisée dans les chambres n'est possible, comme un ancien client revenant après qu'un nouveau est arrivé. Ce problème n'est pas avant tout un problème de codage informatique, mais de spécification du comportement même des clés[81].

Ainsi D. Jackson décrit la programmation comme étant essentiellement un travail de *conception*, qui capture la forme essentielle des idées en jeu dans le problème considéré :

> Le logiciel est construit sur des abstractions. Choisissez les bonnes, et la programmation découlera naturellement de la conception ; les modules auront des interfaces réduites et simples ; et les nouvelles fonctionnalités s'intégreront aux anciennes sans réorganisation en profondeur. Choisissez les mauvaises, et la programmation sera une série de mauvaises surprises : les interfaces deviendront baroques et maladroites car elles seront obligées de s'adapter à des interactions imprévues, et même les changements les plus simples seront difficiles à faire. Nulle réorganisation, aussi ample soit-elle, ne peut sauver un système construit sur des concepts défectueux – à part recommencer à zéro. […]
>
> *Le cœur du développement logiciel est donc la conception d'abstractions*. Une abstraction n'est pas un module, ni une interface, une classe ou une méthode ; c'est une structure pure et simple, une idée réduite à sa forme essentielle. Puisqu'une

79. (Abrial 2013, p. 2).
80. (D. Jackson 2021, p. 196).
81. (D. Jackson 2006, p. 191).

> même idée peut être réduite à des formes différentes, les abstractions sont toujours, en un sens, des inventions, même si les idées qu'elles réduisent existaient auparavant dans le monde extérieur au logiciel. Les meilleures abstractions, cependant, capturent leurs idées sous-jacentes de manière si naturelle et convaincante qu'elles ressemblent davantage à des découvertes[82]. *(Nous soulignons)*

D. Jackson argumente en conséquence que les méthodes formelles sont un moyen d'analyser ces abstractions et de s'assurer de leur robustesse *avant* de commencer le travail de codage. D'une certaine manière, l'essentiel de la programmation se fait là, dans ce pur travail de spécification du problème.

§ 201. L'invention conceptuelle et le design

Cette définition de la programmation trouve de nombreux échos chez les professionnels, auxquels il apparaît très vite que les bonnes heuristiques sont indissolublement liées à la bonne conception du problème.

Par exemple, les schémas conceptuels (*design patterns*) désignent des cadres conceptuels, rassemblés dans quelques ouvrages de référence[83], et enseignées dans les formations informatiques, qui recensent des situations et problèmes typiques de programmation, ainsi que des manières générales de les aborder. Une illustration en est le schéma Modèle-Vue-Contrôleur (MVC) utilisé dans le développement d'interfaces entre une application et ses utilisateurs. Ce schéma recommande de décomposer la partie du programme concernant l'interface en trois modules : le modèle est l'application proprement dite, qui notamment maintient à jour toutes les données critiques. Les contrôleurs sont les procédures par lesquelles l'utilisateur peut interagir avec elle (par la souris, le clavier, la caméra...), et les vues sont les procédures de présentation de l'application, typiquement des fenêtres de graphiques et de texte, mais également des sons ou des informations envoyées aux périphériques. Le découplage de ces trois éléments permet d'améliorer grandement la robustesse et l'évolutivité de l'application.

Il y a également conceptualisation au niveau du codage proprement dit, puisque les procédures sont construites sur la base de représentations structurées de l'information et en sont étroitement dépendantes. Fred Brooks, dont l'ouvrage *The mythical man-month* est devenu un classique du développement logiciel,

82. (D. Jackson 2006, p. 1).
83. Le texte fondateur est (Gamma et al. 1994).

affirme comme Jackson, que la conception d'abstractions y est centrale :

> *La représentation est l'essence de la programmation.* Au-delà du savoir-faire se trouve l'invention, et c'est là que naissent les programmes sobres, simples et rapides. Ils sont presque toujours les résultats d'une percée stratégique plutôt que d'une intelligence tactique. Parfois, il s'agira d'un nouvel algorithme, tel que la transformation de Fourier rapide de Cooley-Tukey. Mais bien plus souvent, *cela viendra d'un retravail de la représentation des données ou des tables*. C'est là que réside le cœur d'un programme. Montrez-moi vos diagrammes de flux* et cachez vos tables, et je continuerai à être mystifié. Montrez-moi vos tables, et je n'aurai généralement pas besoin de vos diagrammes de flux* ; ils seront évidents[84]. *(Nous soulignons)*

C'est également ce que dit Dijkstra :

> C'est ce qu'un programmeur doit faire tout le temps : il doit introduire de nouveaux concepts – qui n'apparaissent pas dans l'énoncé du problème d'origine – afin de pouvoir trouver, décrire et comprendre sa propre solution du problème. […] Étant donné le problème, le programmeur doit développer (et formuler !) la théorie nécessaire pour justifier son algorithme[85].

Nous revenons plus loin sur cette expression de *théorie* qui vient naturellement à Dijkstra lorsqu'il évoque la résolution de problème. Comme exemple d'innovation conceptuelle, nous mentionnons seulement l'exemple des types abstraits de données (*abstract data types*), dus à Liskov et Zilles en 1972[86], car cette idée nous semble capturer une dimension essentielle de ce que sont les moyens matériels, tels que nous les analysons suite à Dewey.

Très simplement[87], l'idée reprend ce que dit déjà Dewey des objets algébriques dans sa *Logique*[88], que l'élément d'un système mathématique est défini par les opérations seules qui peuvent le solliciter. Par exemple un entier est défini par les opérations d'addition, soustraction, multiplication, et division et par les relations d'ordre (plus grand ou plus petit que), toutes régies par des axiomes (associativité, commutativité, etc.). La manière dont les entiers sont représentés par l'ordinateur n'est pas accessible à l'utilisateur et ne doit pas avoir d'importance pour lui. Ou encore une pile* d'informations est définie par trois opérations d'ajout, d'extraction et de lecture, qui opèrent toutes trois sur l'élément supérieur de la pile. La manière dont la pile est elle-même représentée dans la mémoire n'est pas acces-

84. (Brooks 1982, p. 102).
85. (Dijkstra 1974, p. 611).
86. Pour l'histoire de cette idée, voir (Liskov 1996).
87. Voir sur Wikipedia, l'article *Abstract data type*.
88. Voir plus haut § 177.

sible à l'utilisateur, qui ne peut donc, par exemple, savoir quel élément se situe au bas de la pile sans en extraire tous les éléments l'un après l'autre.

Cette idée, que les données ou informations doivent être spécifiées par leurs opérations afin qu'un système de significations acquière son autonomie logique à l'égard du système sous-jacent, peut nous paraître intuitive d'une certaine manière. Elle correspond exactement à la conception de Dewey concernant les systèmes mathématiques, qu'ils doivent fournir des moyens matériels définis entièrement par des opérations possibles sur eux. Les types de données abstraits furent cependant à l'époque une réelle conquête intellectuelle qui mûrit lentement à la convergence de plusieurs recherches indépendantes, d'une part concernant la modularité des programmes, et d'autre part la vérification, qui exigeait une organisation hiérarchique de l'accès aux données.

Dans son dernier ouvrage, au titre ambitieux *The Essence of Software* (l'essence du logiciel), D. Jackson reprend son idée manifeste, que la programmation a trait à la conception, en la rapprochant de celle de *design* dans son sens courant, et en se plaçant ainsi dans la perspective fondamentale d'Herbert Simon. Il propose de mettre en avant la notion de *concept* qu'il définit par une intention (*purpose*), des états (*states*), des actions, et des contraintes internes (*operational principles*). L'exemple suivant définit de manière semi-formelle le concept de liste de tâches (*todo*) :

Code 11.1 – Description d'un concept « todo »

```
concept todo
purpose keep track of tasks
state
    done, pending: set Task
actions
    add (t: Task)
        when t not in done or pending
        add t to pending
    delete (t: Task)
        when t in done or pending
        remove t from done and pending
    complete (t: Task)
        when t in pending
        move t from pending to done
operational principle
    after add (t) until delete (t) or complete (t), t in pending
    after complete (t) until delete (t), t in done
```

La définition du concept `todo` (« liste de courses ») procède par la définition de deux états possibles des tâches (`Task`) : `done` (« fait ») et `pending` (« à faire ») (lignes 2-3), puis par trois actions qu'il est possible de réaliser sur des tâches individuelles de la liste (lignes 4 - 13) `add, delete, complete` (« ajouter, effacer, terminer ») et dont la définition est chaque fois une règle introduite par une condition (`when`). Le point essentiel du concept sont ses deux principes opérationnels, définis lignes 14-16, qui sont des contraintes que les tâches de la liste doivent satisfaire. Ces contraintes mettent en relation états et actions. Le premier principe (ligne 15) dit qu'une tâche qui n'est ni terminée ni effacée est dans l'état *à faire*, et le second (ligne 16) qu'une tâche terminée est *faite*.

On peut reconnaître dans cette notion des similarités avec celle de type abstrait de données, mais elle en diffère par deux aspects essentiels selon D. Jackson[89] : d'abord, un concept se situe sur le plan du problème lui-même et non sur celui de la machine. Il ne s'agit pas d'une notion informatique au sens courant du terme, ce qui se voit à la mention liminaire de son intention, c'est-à-dire du problème qu'il vise à résoudre. Ensuite, et de manière plus novatrice, Jackson insiste sur l'indépendance relative des concepts vis-à-vis du système de significations dans lesquels ils prennent place. On objectera que, dans l'exemple cité, le concept de liste de tâches (*todo*) semble dépendre au moins de celui de tâche (*task*, ligne 4). Or justement D. Jackson insiste sur le fait que ce dernier concept *reste indéterminé* dans cette définition, ce qui fait que le concept de liste de tâches pourrait être utilisé dans des contextes très différents, comme celui de courses alimentaires, de travaux de bricolage, d'obligations administratives, etc. qui définiraient chaque fois des concepts spécifiques de tâche. Il est banal de dire que tout concept ne prend de signification qu'au sein d'un système de concepts ; D. Jackson s'inspire de l'idée informatique de liaison tardive* (*late binding*) pour montrer que, si tel est bien le cas chaque fois qu'on veut résoudre la signification d'un concept dans un jugement concret ayant lieu dans un contexte particulier, c'est bien l'indétermination générique du concept qui en fait sa polyvalence. Le projet de D. Jackson, explicité sur un site qu'il lui consacre[90], est ainsi de créer une bibliothèque universelle de concepts techniques, définis de manière robuste par des intentions chaque fois uniques, et qui pourraient être assemblés « à la volée » en un système de signifi-

89. (D. Jackson 2021, p. 246-247).
90. forum.softwareconcepts.io.

cations adapté au problème à résoudre, ce que Jackson appelle une application (*app*).

Cette notion originale de *concept* nous semble expliciter et préciser l'inspiration originale de Kant, que nous avons suivie aussi bien chez Dewey que chez Martin-Löf, et qui le définit comme *règle*, et entrer également en résonance avec les recherches psychologiques contemporaines qu'Osta Vélez (2020) appelle les doctrines inférentialistes du concept[91]. Cela milite pour l'expansion de notre compréhension de la programmation, non seulement comme d'une discipline technique ou même mathématique, mais comme une entreprise d'invention conceptuelle et de clarification des concepts au service de l'investigation et de la résolution de problème.

Comme l'algorithmique, la discipline théorique des méthodes formelles, dont nous sommes partis, s'inscrit donc dans une dimension pratique plus large de la programmation, complémentaire à l'heuristique, qui est celle du *design* ou de la conception orientée vers la résolution de problème. De la même manière que l'heuristique et l'algorithmique cherchent à développer et vérifier les moyens procéduraux de systèmes mathématiques, les méthodes formelles et le *design* cherchent à développer et vérifier leurs moyens matériels. Si la programmation est bien partie de ces mathématiques constructives, telles que nous les avons caractérisées plus haut, il est logique qu'on l'associe plus naturellement à sa dimension procédurale. Cependant, en tant qu'elle est mobilisée de manière universelle pour formaliser les résultats d'une investigation, elle a également affaire de manière essentielle à l'établissement de systèmes mathématiques de significations. Sa particularité est d'avoir un champ bien plus large que celui dévolu historiquement à la logique, puisque les problèmes auxquels elle a affaire ne concernent pas seulement les jugements scientifiques, mais également ceux du droit, de l'administration, des affaires, et plus généralement encore, tous les problèmes de l'action humaine rationnelle, qu'elle soit technique, économique, ou institutionnelle.

11.3 Qu'est-ce qu'une machine ?

L'interprétation de la programmation comme mise en procédure systématique d'une investigation pourrait être soutenue, avec des formules différentes, par les théoriciens de l'informatique déjà cités comme Hoare et Dijkstra. Une objection

91. Voir plus haut, § 132.

centrale, cependant, est qu'elle élude complètement le fait que la programmation a principalement affaire à des machines bien réelles, les ordinateurs.

C'est à répondre à cette objection que sont consacrées les trois sections suivantes, c'est-à-dire à comprendre quel rôle y joue le concept de *machine* – un terme à peine cité à la section précédente. Il est apparu central dans notre circonscription initiale de la programmation, qui l'a placée au centre d'une *négociation* de significations entre le maître d'ouvrage et le programmeur. Ces réflexions permettront, à la fin du chapitre, de faire droit à la première objection et d'étendre de manière significative notre interprétation de la programmation.

Nous commençons par examiner plusieurs critiques faites à la conception mathématique de la programmation développée aux sections précédentes, sans les juger décisives. Nous prêtons par contre une grande attention à l'emploi, semble-t-il métaphorique, du terme *mécanique* par les théoriciens de l'informatique pour désigner la qualité essentielle requise de l'exécution, l'*absence totale de jugement*, hormis celui qui permet de reconnaître les signes pointant sur les opérations à réaliser (§ 202). Cela nous amène à considérer les approches *mathématiques* de la notion de *machine*, qui permettent de la voir comme un « micro-espace » de systématicité, un lieu où des règles peuvent être appliquées exactement ; et c'est cela qui rend la machine *homogène* aux procédures pures produites par l'investigation (§ 203). Cela nous amène à en proposer une définition nouvelle qui, nous semble-t-il, fait droit à la polysémie très ancienne du terme (§ 204). À la section suivante, nous précisons cette approche en identifiant les ordinateurs *aux contrôles de transmission* des machines.

§ 202. Qu'est-ce qu'une conduite « mécanique » ?

Les développements précédents pourraient apparaître comme incorrigiblement abstraits aux praticiens de l'informatique, qui connaissent bien les difficultés d'une implémentation réussie[92]. Même s'il ne s'agit pas toujours de questions « matérielles » (vitesse, mémoire, latence, etc.) tout programme relativement complexe doit tenir compte des contraintes imposées par le système sous-jacent, dont nous avons déjà tenté de donner un aperçu[93].

Cependant, comme nous l'avions vu, un tel argument ne remet pas en cause

92. Pour une revue des relations historiques entre mathématiques et informatique, voir (Tedre 2014, ch. 3 et 4).
93. Voir plus haut, section 3.2.

les méthodes formelles, bien au contraire : à chaque niveau d'implémentation une approche formelle est possible, voire souhaitable, afin de décrire adéquatement ce dont il est question. À chaque niveau, des concepts, des algorithmes, des heuristiques et surtout de la logique sont nécessaires pour écrire des programmes de bonne qualité, et protester qu'il y a toujours un « je ne sais quoi », qui échappe à l'analyse rationnelle, pointe plus vers la difficulté à réaliser le travail formidable de la spécification que vers son inutilité.

On pourrait également objecter que la notion de *machine* est bien présente de manière subreptice dans les méthodes formelles, à travers celles de *langage machine* et de *compilateur* que nous avons utilisées à propos des sémantiques opérationnelle et dénotationnelle. Vérifier la correction d'un langage de programmation semble en effet requérir de se référer au fonctionnement d'une machine, même si celle-ci est seulement stylisée par une fonction de transition d'états. Par ailleurs, l'étude des algorithmes requiert que soit définies des opérations élémentaires afin que leur complexité puisse être mesurée : là aussi, des machines peuvent être entrevues en filigrane[94]. On notera d'ailleurs que, si la thèse de Church-Turing affirme que tous les modèles de machines d'un certain niveau de complexité permettent de calculer les mêmes fonctions, l'expression des algorithmes peut différer radicalement d'une machine à l'autre.

Cependant un tel argument est fragile, et objet de débat. D'autres propositions, comme celles de Gurevich et Spielmann (1997) ou Moschovakis (2001), tentent de les définir de manière indépendante de tels modèles, par le biais de structures mathématiques qui capturent directement leur signification. Cela est également la direction que prennent les recherches de Naibo (2022). Ces directions de recherche nous semblent montrer que la machine désigne ici la modalité spécifique *de parcours systématique de l'espace de problème*, notamment par le biais de la notion de récursivité, dont nous avons vu le caractère central. La pragmatique de Caracciolo ne pointe donc pas nécessairement vers les modalités empiriques de l'implémentation d'une structure mathématique ; elle peut être aussi interprétée formellement, comme s'intéressant à la mesure du temps de parcours de l'espace de problème. D'ailleurs, pour exposer cette notion, Caracciolo remplace l'idée de machine par celle plus neutre et plus stylisée d'un simple *exécutant*[95].

Pour comprendre l'importance de la machine dans les fondements conceptuels

94. Voir (Van Leeuwen 1990, vol. A, p. 19), (Dean 2016, p. 23).
95. (Voir plus haut p. 763.

de l'informatique, il ne faut donc pas chercher sa trace dans l'épistémologie des algorithmes ou des méthodes formelles, mais revenir à la caractéristique principale que le programmeur demande à tout exécutant qui devrait appliquer sa procédure, à savoir qu'il se comporte *mécaniquement*. Une telle expression est tout autant mystérieuse qu'elle est récurrente dans les textes fondamentaux de la théorie de la calculabilité, depuis l'analyse célèbre de Turing (« On peut comparer un homme en train de calculer un nombre réel à une machine qui n'est capable que d'un nombre fini de conditions[96]... ») qui la présente, à strictement parler, comme une métaphore. C'est en particulier Gödel, à partir de sa *Gibbs Lecture* de 1951 qui insiste sur l'importance de la définition *mécaniste* par Turing de la notion de procédure effective :

> La manière la plus satisfaisante... [d'arriver à une telle définition] est de réduire le concept de procédure finie à celui d'une machine ayant un nombre fini de pièces, comme l'a fait le mathématicien britannique Turing[97].

Avant cela en effet, lors de la conférence de Princeton de 1946 qu'il avait organisée sur le sujet, Tarski avait formulé le problème de la décision de la manière suivante :

> Matériellement, nous pouvons justifier le choix du problème de la décision par le fait que la tâche de la logique est de mécaniser la pensée […] De manière intuitive, la solution du problème de la décision consiste à déterminer s'il existe un moyen mécanique de décider si un énoncé donné d'un système formel est un théorème[98].

Pourtant c'est bien chez Gödel qu'on trouve l'une des premières occurrences de cette idée, dans un texte de 1933 :

> [Les règles d'inférence] se réfèrent uniquement à la structure extérieure des formules, et non à leur signification, de sorte qu'elles puissent être appliquées par quelqu'un qui ne connaîtrait rien aux mathématiques, *ou par une machine*. Cela a pour conséquence qu'il ne peut jamais y avoir de doute quant aux cas auxquels les règles d'inférence s'appliquent, et qu'ainsi on obtient le degré d'exactitude le plus élevé possible[99]. *(Nous soulignons)*

Dans ce texte précurseur, l'image de l'extériorité des règles au contenu des formules symboliques amène celle de la machine, comme d'un dispositif *fermé* à toute

96. (Turing 1937, p. 231).
97. (Collected Works III, pp. 304-305), cité par (Sieg 2006, p. 196). Nous nous appuyons sur ce dernier article concernant cette remarque historique sur Gödel.
98. Texte publié par (Bénis-Sinaceur 2000, p. 23, 24). Tarsksi utilise également le concept de machine dans un texte de 1939, également cité par (ibid., p. 8).
99. Texte de 1933, cité par Sieg (2006, p. 197).

compréhension. Ce raisonnement est entièrement explicité par Minsky, que nous citons donc dans sa totalité :

> Nous commencerons par dire qu'une procédure effective est un ensemble de règles qui disent, d'instant en instant, précisément comment se comporter.
> Cette tentative de définition est sujette à la critique selon laquelle l'interprétation des règles est laissée à la charge d'une personne ou d'un agent. Or la capacité d'une personne à obéir à des instructions dépend de ses antécédents et de son intelligence. Si son intelligence est trop faible, il pourrait ne pas comprendre ce que nous voulons dire. Si son intelligence est trop grande ou trop étrangère, il pourrait inventer une interprétation cohérente des règles qui n'était pas prévue. On sait combien de fois une explication apparemment « simple » se révèle porteuse d'une ambiguïté insoupçonnée.
> Nous pourrions éviter les problèmes d'interprétation de la compréhension si nous pouvions préciser, avec l'énoncé des règles, les détails du *mécanisme* qui doit les interpréter. Cela ne laisserait aucune ambiguïté. Bien sûr, il serait très fastidieux de devoir refaire tout cela pour chaque procédure individuelle ; il est souhaitable de trouver une famille raisonnablement uniforme de mécanismes obéissant aux règles. La situation la plus pratique serait celle dans laquelle nous mettrions en place
> (1) un langage dans lequel des groupes de règles de comportement seraient exprimées, et
> (2) *une machine unique* qui pourrait interpréter les instructions dans ce langage et ainsi exécuter les étapes de chaque processus spécifié[100]. *(Nous soulignons)*

Dans ce texte remarquable, on voit que c'est le problème de l'interprétation et de la communication interpersonnelle qui suscite l'appel au mécanisme et à la machine de manière à évacuer tout ambiguïté, comme le « *Calculemus !* » leibnizien en son temps. Mais alors que le calcul leibnizien devait se fonder sur une langue et des caractères véhiculant immédiatement les significations primitives des choses, ici au contraire, c'est en vidant les symboles de leur sens, et les agents de leur capacité de compréhension, qu'on assure la communication.

Enfin, cette exigence se rencontre dans les nombreuses définitions *mathématiques* des notions de programme, de procédure ou d'algorithme, comme celle-ci donnée par Knuth :

> Il existe plusieurs autres mots qui capturent presque, mais pas tout à fait, le concept nécessaire : procédure, recette, processus, routine, méthode. Comme ces choses, un algorithme est un ensemble de règles ou de directives visant à obtenir un résultat

100. (Minsky 1967, p. 106).

> spécifique à partir d'une donnée spécifique. La particularité d'un algorithme est que tout flou doit être éliminé ; les règles doivent décrire des opérations si simples et si bien définies qu'elles puissent être exécutées par une machine[101].

Tous ces textes ont pour point commun de présenter la machine comme un certain idéal-limite de l'exécution humaine de règles. Comment interpréter cela ? On peut l'interpréter *littéralement*, comme le fait Hersh lorsque, dans sa croisade antiformaliste, il va jusqu'à réduire l'essence des preuves formelles à leur seule calculabilité mécanique :

> Qu'est-ce que la logique ? S'agit-il des règles de la pensée correcte ? L'expérience quotidienne et de nombreuses études menées par des psychologues montrent que la plupart de nos pensées ne suivent pas la logique. […] Les machines à calculer obéissent presque toujours à la logique. C'est la réponse ! La logique donne les règles des machines informatiques ! La logique s'applique aussi aux personnes lorsqu'elles essaient d'être des machines informatiques[102].

Cependant cette interprétation littérale n'est pas compatible avec le fait historique que l'idée très ancienne de règles logiques n'est jamais rapportée à celle de machine avant les années 1930.

Pour comprendre la liaison entre procédure et machine, il faut revenir à son inspiration hilbertienne, selon le mot de Bénis-Sinaceur[103]. Elle s'inscrit en effet dans le projet de définir les critères que doivent satisfaire les règles de l'inférence mathématique, afin qu'elles fassent aussi peu appel que possible à l'intelligence ou à l'intuition du mathématicien – puisque c'est à ce propos que des désaccords étaient survenus concernant la question de l'infini. Le calcul est pris par Hilbert comme paradigme de règles exactes, et il propose donc pour critère que l'inférence soit définie par des règles symboliques finitaires, c'est-à-dire qui ne procèdent que par reconnaissance et substitution de symboles d'une proposition à l'autre, en un nombre fini d'étapes. La machine apparaît donc, dans cette perspective comme un *idéal-limite* du calculateur humain, idéal qui exprime la certitude que celui-ci ne fait qu'obéir à des règles parfaitement comprises, donc dénuées de toute ambiguïté, sans exhiber d'autre forme d'intelligence et d'intuition.

Un tel rapport évoque la remarque de Galilée que la machine permettait d'économiser le jugement de l'ouvrier[104]. Le calcul est l'une de ces tâches subalternes où

101. (Knuth 1977, p. 63).
102. (Hersh 1997, p. 140).
103. (Bénis-Sinaceur 2000, p. 7).
104. Voir plus haut, p. 470.

le jugement n'est pas nécessaire, voire néfaste à l'accomplissement de ses résultats. Babbage a l'idée de sa machine différentielle car il veut économiser, tout autant que le salaire de ses *computers*, son propre temps passé à vérifier leurs calculs[105].

La machine n'automatise donc le jugement humain qu'en le supprimant entièrement. Dans le cadre de la théorie de la connaissance de Dewey, cela signifie que la procédure, telle que la résolution finale du problème la reconstruit, permet de *se passer entièrement* de l'investigation. Cela n'est possible, avons-nous dit, que pour les investigations entièrement procédurales (mathématiques) établies sur des systèmes de signification préétablis. On voit bien ce qu'il y a d'intuitif dans une telle possibilité : lorsqu'on cherche une solution *systématiquement*, c'est-à-dire en parcourant un espace de problème selon des règles préétablies, il est crucial de « ne pas sauter d'étapes » et de ne pas laisser l'intuition reprendre ses droits.

Cependant une telle exigence explique le recours métaphorique à l'image de la machine, mais non pas la possibilité qu'il y a à transférer de manière effective les opérations prescrites par la procédure, qui se réfèrent à des actions possibles pour le sujet humain, à des significations décrivant le comportement possible de machines.

Autrement dit, la question qui se pose à présent est celle-ci : pourquoi la reconstruction *systématique* d'une investigation, la procédure, se laisse-t-elle transférer dans le système de significations particulier qui décrit le comportement mécanique (ou électronique) des machines ? Nous avons déjà maintes fois affirmé la proximité des notions de systématicité et de machines, sans jamais pourtant l'aborder aussi directement. Quel rapport essentiel peut-il y avoir, en effet, entre une procédure qui permet de parvenir à une décision *humaine* et une machine, artefact matériel ? Cette difficulté n'est rien d'autre que la formulation du paradoxe populaire associé aux expressions de *machines pensantes* ou d'*intelligence artificielle*. On perçoit intuitivement le chiasme qui existe entre, d'une part, les règles de conduite que l'on s'impose à soi-même ou que l'on convient avec autrui, et d'autre part le comportement régulier d'objets mécaniques.

Répondre qu'on construit les machines afin qu'elles aient cette propriété, d'obéir à des règles, ne fait que repousser la question. Car comment les construit-on pour qu'elles intègrent ces règles ? On dira qu'on utilise leurs propriétés naturelles, entendant par là les lois naturelles auxquelles elles se conforment, et notamment les lois de la mécanique. Ainsi, la machine du levier

105. (Babbage [1822] 1989a, p. 15).

utilise la loi du moment mécanique pour multiplier une force, le moulin concentre une force en un point d'application, etc.

Il n'y a cependant dans ces machines pas d'autre règle que celles de transmission des forces, mais rien qui ressemble à une règle de conduite. Il n'est ainsi pas du tout évident que des procédures, établies par l'investigation raisonnable, puissent ainsi être transférées à un objet matériel. Piccinini (2015) expose bien la difficulté :

> Si vous pensez que la théorie de la calculabilité concerne des objets abstraits et qu'il existe de telles choses – c'est-à-dire si vous êtes platonicien – vous devez vous demander : que faut-il à un système physique concret donné pour qu'il implémente un objet computationnel abstrait donné (par opposition à un autre objet abstrait, ou aucun) ? […] Si vous pensez qu'il n'y a pas d'objets abstraits, vous devez vous demander : que faut-il à un système physique concret donné pour qu'il satisfasse une description computationnelle abstraite donnée (par opposition à une autre, ou aucune)[106] ?

La question qu'il convient de se poser n'est donc pas seulement : « *Pourquoi* la programmation est-elle programmation de *machines* ? », mais plus radicalement encore : « *Comment est-il possible* que la programmation soit programmation de *machines* ? »

§ 203. La théorie des machines

Cette question requiert d'élaborer à neuf la notion de *machine* et de voir *ce qu'il y a de mathématique en elle*. C'est ainsi qu'Ashby, un des fondateurs de la cybernétique, recouvre l'idée d'une théorie des machines qui serait similaire à la géométrie :

> De nombreux ouvrages ont porté le titre « Théorie des Machines », mais ils contiennent habituellement des informations sur des choses *mécaniques*, commes des leviers ou des engrenages. La cybernétique également, est une « théorie des machines », cependant elle ne traite pas de choses, mais de *manières de se comporter*. […] La cybernétique a rapport avec la machine réelle – électronique, mécanique, neurale, ou économique – comme la géométrie a rapport aux objets dans notre espace terrestre. […]
>
> Une machine déterminée est définie comme ce qui se comporte de la même manière qu'une transformation close univalente[107].

106. (Piccinini 2015, p. 7).
107. (Ashby [1956] 1957, p. 1-2, 24).

Par cette dernière expression Ashby entend une fonction totale d'un ensemble d'états sur lui-même, c'est-à-dire rien d'autre qu'une fonction de transition. On voit très clairement dans la perspective d'Ashby qu'une machine ne signifie pas seulement ce qu'on entend habituellement par là – un artefact industriel – mais tout objet, naturel ou artificiel, physique, animé ou humain qu'on peut décrire par une telle fonction. La machine est donc ici une catégorie descriptive du réel, qui peut être appliquée à tout phénomène pour le comportement duquel on peut exhiber une telle fonction.

Cette idée est cristallisée dans le concept de *calcul physique* (*physical computation*), qui a été l'objet de nombreuses analyses philosophiques. En particulier, Gandy (1980) et Sieg (2009) soutiennent la thèse que les *machines* sont, dans leur nature la plus générale, les phénomènes qui peuvent être décrits par des systèmes dynamiques discrets satisfaisant certaines conditions de finitude et de cohérence. Cette description est mathématique. D'une part l'état du système est une structure enchâssée, sur plusieurs niveaux (en nombre fini) d'éléments quelconques, appelés atomes. D'autre part l'évolution dynamique du système est décrite comme une fonction modifiant cette structure seulement localement. Elle peut néanmoins y ajouter de nouveaux atomes, la dynamique pouvant donc être indéfiniment expansive ; par ailleurs si des modifications locales viennent à interagir, elles sont « coordonnées » par cette fonction[108].

En somme, les interprétations d'Ashby et de Gandy, malgré leurs différences, ont en commun d'identifier les machines aux systèmes phénoménaux dont la description satisfait certaines contraintes procédurales, qu'ils soient naturels ou artificiels. Cependant ces caractérisations sont faiblement opérantes pour notre problème. Il ne s'agit pas de savoir si on peut reconnaître un calcul dans le comportement d'un système, mais comment il se fait que nous puissions *programmer* des systèmes pour qu'ils réalisent les calculs que nous souhaitons leur déléguer. On pourrait être en effet dans la situation où nous pourrions reconnaître, dans le monde, de nombreux phénomènes naturels répondant à tous nos besoins calculatoires, sans jamais que nous eussions à en programmer un seul, à construire une seule de ces machines.

Minsky (1967, p. 1-6), dans sa réflexion sur les machines, se situe globalement dans la même ligne. Il ajoute cependant au terme de son développement une remarque intéressante concernant la particularité des machines comme *artefacts*.

108. (Sieg 2009, p. 589-591).

Leur mécanisme n'est pas une *description* mais une *spécification* de leur fonctionnement. Contrairement à une théorie, s'il y a désaccord entre la représentation du mécanisme et la machine, c'est la machine qu'il faut remplacer. Aussi, ajoute Minsky – et là se trouve la réflexion importante :

> Comment pouvons-nous être sûrs, dans un cas quelconque, qu'il existe un moyen de construire la machine que nous avons à l'esprit ? Comment savons-nous que le monde physique le permettra ? Dans certains cas particuliers, aucune personne raisonnable n'aura de doute là-dessus. Puis-je vraiment construire une machine à additionner ? Lorsque nous regardons la structure d'une simple machine à additionner, nous voyons qu'elle devrait fonctionner. Elle ne peut pas ne pas fonctionner ! – cela semble aussi clair que 2 + 2 = 4.
>
> Le cliché est éclairant. Certaines personnes croient que de simples énoncés mathématiques sont « évidents ». D'autres soutiennent qu'ils sont basés sur des observations encore obscures mais bien empiriques. L'un ou l'autre point de vue peut être tenu pour les machines. Qu'une machine à additionner conventionnelle fonctionne (si elle est correctement construite) semble d'une part équivalent aux propositions arithmétiques impliquées, et d'autre part semble dérivé de notre expérience empirique sur la manière dont les objets se comportent dans le monde. *Peut-être pourrait-on même soutenir l'idée que la croyance en un énoncé arithmétique équivaut à celle que certaines machines, correctement construites, fonctionneront*[109].

La question que pose Minsky au début de ce texte est celle-là même qui guide notre questionnement : comment est-il possible d'implémenter des procédures « mentales » dans des dispositifs « concrets » ? Il suggère, avec grande finesse, que la réponse pourrait se situer dans la *superposition* de deux régimes d'argumentation : un régime empirique, qui doit assurer que la machine concrète répond à la spécification, et un régime logico-mathématique, où cette dernière n'est rien d'autre qu'une reformulation de théorèmes mathématiques, c'est-à-dire l'épreuve de la cohérence avec soi-même[110]. Une machine fonctionne *parce qu'elle est construite* pour fonctionner ainsi, de la même manière qu'en logique constructive, un jugement est entièrement constitué par la preuve qui permet de l'affirmer. Mais par ailleurs, toutes les descriptions des éléments constitutifs de la machine, à partir desquels elle est construite, sont *supposées empiriquement* être valides.

La superposition des arguments empirique et mathématique est suggérée dans la dernière phrase du texte. L'équivalence des deux croyances est une double im-

109. (Minsky 1967, p. 5). Les italiques sont de Minsky.
110. Voir plus haut, § 149.

plication : d'une part, il y a l'implication « platoniste », que la machine fonctionne parce que l'énoncé mathématique est correct, mais d'autre part aussi l'implication inverse, que cet énoncé est correct *parce que la machine, correctement construite, fonctionne*. On croit entendre ici le lointain écho de l'affirmation hégélienne de la réalité de la raison, et de la rationalité du réel, mais cette intuition se comprend plus simplement, à notre avis, dans le cadre épistémologique que nous avons posé. Tout jugement réfléchi sur une réalité phénoménale est à la fois la construction d'une « machinerie logique », si l'on peut dire, et l'assertion que celle-ci s'applique à cette réalité. Cette machinerie n'est rien d'autre que l'indication d'une procédure pour agir (empiriquement) sur cette réalité, et la promesse empirique que cette action aboutira au résultat prévu par la procédure.

§ 204. La machine comme objet dynamique réglé

Ce qui ressort de cette première enquête est la difficulté de définir le terme de *machine*. Cette difficulté n'est pas récente : Marx la notait déjà[111]. Séris[112] remarque d'une manière plus générale que l'histoire des machines mécaniques se prête à des reconceptualisations constantes. Face aux machines, on a irrésistiblement tendance, dit-il, à définir leur périmètre et leur signification en fonction de leurs plus récentes évolutions, comme si le progrès technique consistait toujours, au-delà de l'accroissement apparent de leur complexité, à mieux révéler leur essence simple, à partir de laquelle s'éclairerait rétrospectivement le sens de leurs ébauches successives. La notion de machine semble par ailleurs déborder le domaine de la technique. Le Littré de 1873 consacre à peu près la moitié de son grand article sur le sujet[113] à des usages qui ne désignent pas des artefacts matériels. Cette diversité peut être retracée jusque dans l'Antiquité, où la machine, dans cette dualité constante, désignait déjà les dispositifs simples de la mécanique (levier, poulie, etc.) et composés (les machines de guerre et de théâtre), mais aussi toute invention ingénieuse et tout moyen mis en œuvre pour parvenir à ses fins.

Il ne s'agit pas selon nous d'un phénomène lexical accidentel. Dans les développements consacrés à la mécanique[114], nous avons mis en évidence le rôle central de

111. (Marx [1867] 2006, p. 416-417).
112. (Séris 1994b, p. 167-172).
113. (Littré 1873, t. 3, p. 368). 9 entrées sur 21 ne concernent pas des artefacts matériels, celles à partir du n° 12 et sauf le n° 16, qui traite des machines de théâtre.
114. section 7.4 et section 7.5.

la notion de *machine* dans l'articulation du savoir théorique et du savoir pratique. Objet de la géométrie du mouvement, elle représente l'idéal d'une portion de la nature matérielle à laquelle correspondrait une construction géométrique susceptible de démonstration et de savoir certain. Comme le dit Wagner[115], elle est pour Descartes « un modèle explicatif qui se comprend par référence aux machines et qui exprime un idéal d'intelligibilité des phénomènes de la nature », notamment par opposition aux qualités occultes de la physique scolastique, en ce que son explication se réduit toujours clairement à de la géométrie. Aussi, comme le dit également Séris,

> la machine (artificielle) n'a pas de privilège par rapport à la machine (naturelle) : ce serait même plutôt le contraire, elle est *moins* machine que le vivant, par exemple ; et elle s'insère dans un monde (« mécaniste ») où tout est déjà machine[116].

La machine en ce sens générique n'est donc pas d'abord un artefact mais, comme le propose Ashby en particulier, une catégorie, qui peut être mobilisée pour décrire le réel et structurer l'expérience. Nous avions avancé que le concept, chez Kant, pouvait être vu comme une machine inférentielle[117]. Dewey dit que la machine est une méthode[118], en ce qu'elle réalise sa fonction en enchaînant des opérations.

Il faut cependant ici se garder d'une assimilation hâtive, qu'on pourrait appeler anthropomorphique, qui se contenterait d'affirmer que les opérations de la machine sont des opérations que pourraient accomplir des humains, et que l'on confie à l'automatisme de la machine. Cela peut convenir à des cas très simples, comme un bras mécanique. Mais un moteur à explosion accomplit des opérations d'un autre ordre. Il faut donc repartir de plus haut pour donner sens à cette assertion de Dewey.

Il nous semble qu'on peut décrire comme machine *tout objet dynamique dont l'interaction avec son environnement semble réglée*. Cette définition permet de rendre compte de la diversité de ses emplois dans le langage courant aussi bien que savant, sans qu'elle apparaisse comme une simple multiplication de métaphores. La voûte céleste est bien en ce sens une machine naturelle, comme le sont également les cours d'eaux, et plus généralement, tout flux soumis à une déclivité, puisqu'on peut en décrire la régularité. Les cultures et les animaux d'élevage sont aussi des

115. (Wagner 1998, p. 152).
116. (Séris 1994b, p. 153).
117. Voir plus haut, § 142.
118. Voir plus haut, § 184.

machines naturelles, au sens où on peut décrire, à un certain niveau d'abstraction, leur comportement régulier comme production stable de céréales, de fruits, de lait, d'oeufs, etc. – du moment que leur environnement et leur soin obéissent eux-mêmes à certaines règles et procédures. La *machine de l'État* désigne habituellement l'ensemble des organes administratifs de gouvernement d'un pays, en tant qu'on les perçoit comme fonctionnant de manière coordonnée et réglée. Une intrigue (politique, théâtrale) est une machine en ce qu'elle vise à régenter toutes les interactions signifiantes de ses participants. Pour revenir à la technique, les bio-technologies nous amènent aujourd'hui à qualifier des organismes de synthèse de « machines biologiques », sans que cet emploi sonne comme métaphorique[119]. Enfin, les machines abstraites discutées à la section précédente sont bien des machines à part entière selon notre définition, et non des idées de machines, même si leur dynamique se déroule dans une temporalité logique.

Le principe de fonctionnement d'une machine est appelé *mécanisme*, et ce terme a également un emploi très étendu. Ainsi, on parle du mécanisme d'une pile chimique ou de reproduction d'un virus, alors que ces explications s'appuient sur des notions chimiques ou biologiques plutôt que sur celles décrites par la science physique appelée *mécanique*, et que nous associons aujourd'hui assez largement à la physique newtonienne.

Dans cette perspective, l'idée d'une programmation de machines apparaît moins problématique. Si la machine est une telle catégorie descriptive du réel, elle est donc une fonction logique bien plus qu'un concept empirique désignant un genre particulier d'artefacts. *Programmer n'est rien d'autre que modifier le mécanisme d'un objet de l'expérience qu'on considère être une machine*, lorsque de telles actions de modification sont définies dans le système de significations décrivant l'interaction de la machine avec son environnement, comme par exemple détourner l'eau d'une rivière pour alimenter un entrelacs de canaux d'irrigation. Il s'agit d'une interaction au second degré avec la machine, qui modifie sa loi d'interaction avec son environnement – ici l'eau n'allant plus au fleuve, mais dans le potager.

Exécuter un programme n'est pas mettre en branle un objet inerte, mais intervenir sur un objet « qui tourne déjà ». Cela est possible parce que la machine est (ainsi que nous l'avons définie) un objet dont les conditions du mouvement sont rapportées à son environnement : la machine fera telle opération si tel levier est en-

119. (Bongard et Levin 2021).

clenché, ou telle autre si tel bouton est actionné, etc. – mais laissée à elle-même elle fera tout de même quelque chose (aller au fleuve, dans l'exemple ci-dessus). Le levier, le bouton, etc. ces *commandes* de la machine, sont parties de l'environnement. Certes, certaines machines naturelles n'ont pas de condition d'environnement (le mouvement des planètes), et d'autres ont des conditions qui ne sont pas en notre pouvoir (les marées). Aussi est-ce bien lorsque nous pouvons intervenir sur certaines conditions de l'environnement réglant un phénomène dynamique que nous attribuons à ce dernier le plus volontiers le qualificatif de *machine* – lorsqu'il a des *degrés de liberté*, qui le rendent capable de *plusieurs* comportements ayant un intérêt pour nous.

Ces degrés de liberté sont la plupart du temps discrets – sur une machine classique, on trouve des boutons à deux positions, ou des leviers à 3, 4 positions – ou bien ils peuvent être parcourus de manière exhaustive, comme un variateur analogique qui peut prendre n'importe quelle valeur dans un intervalle donné. Autrement dit ces degrés de liberté peuvent être décrits systématiquement, et donc également (par définition) la variation des règles de fonctionnement de la machine, ainsi que l'ensemble de ses comportements possibles. La machine est un espace potentiel de systématicité.

Nous ne voulons pas dire par là que *toute* conception de machine procède d'une démarche systématique qui se représente clairement toutes les règles de son fonctionnement et les conditions associées à leurs variations possibles. Les machines sont souvent « bricolées » – on se rappelle cette assertion de Koyré, qu'avant l'avènement de l'esprit théorique moderne, les machines des artisans médiévaux appartenaient au « monde de l'à-peu-près »[120]. Par contre, toute machine qui fonctionne, même approximativement, invite, comme le disait également Koyré, à être rendue *fiable* : elle invite à ce qu'on explore systématiquement son mécanisme.

Il faut également souligner qu'il n'est pas nécessaire de connaître *toutes* les lois d'interaction d'un objet avec son environnement pour le traiter comme une machine, mais seulement celles du phénomène qui est d'intérêt pour nous. Ainsi, je peux ne pas comprendre les raisons du comportement d'une poule en général (par exemple pourquoi et comment elle communique avec ses congénères), mais savoir très bien quelles sont les conditions nécessaires à ce qu'elle produise telle quantité d'œufs pendant une période donnée. La poule est dans ce cas machine relativement à la ponte d'œufs, et non à son comportement en général.

120. Voir plus haut, § 122.

De la même manière, je peux ne pas comprendre *exactement* ce qui se passe à l'intérieur du tambour du lave-linge, qui remue le linge et l'eau selon des trajectoires chaotiques ; mais je sais que, quelles que soient ces trajectoires précises, le linge en sortira nettoyé, car cette transformation dépend seulement de l'intensité de ce remuement, de la température de l'eau, du savon, etc.

De nombreux phénomènes dynamiques échappent à ces conditions. Lorsque l'interaction du phénomène avec son environnement est complexe, et que son jeu de conditions varie lui-même au cours du temps, nous ne pouvons pas dégager ses lois d'interaction. Ainsi, lorsqu'il faut décrire des conditions climatiques avec une précision indéfiniment grande pour prévoir l'évolution de la météo à un endroit donné, il y a là un phénomène dont les règles ne peuvent être décrites systématiquement. Nous parlons alors de *système ouvert*, ou encore de *système complexe, chaotique* ou *dynamique*, plutôt que de machine. Tous ces termes, qui se rapportent à la *théorie des systèmes* introduite par Bertalanffy, sont assez révélateurs[121]. Un système ouvert désigne un phénomène qui n'est pas une machine, car nous peinons à décrire systématiquement son comportement : il est un *problème*, un sujet d'étude pour la systématisation. C'est là, d'ailleurs, que la programmation joue un rôle puissant – nous y revenons au chapitre suivant.

Une machine est donc programmable dès qu'il est possible d'agir sur son environnement pour modifier son comportement de manière réglée. Cette modification n'est pas difficile à concevoir. Il suffit par exemple d'un interrupteur qui bloque le mouvement, ou l'oriente dans une direction plutôt qu'une autre. Il s'agit bien d'une opération au second degré, dans le sens qu'elle modifie le réseau d'inférences que l'on est autorisé à faire concernant le mécanisme, c'est-à-dire le fonctionnement de la machine. Programmer, c'est ainsi se représenter à l'avance la séquence d'actions qui opèreront sur le jeu de la machine : on retrouve ici notre définition de la programmation donnée au début de notre réflexion, comme d'« *Un arrangement de signes visant à déterminer le comportement d'une machine* »[122].

La programmation de machines, dans cette perspective, précède donc bien l'informatique. Nous venons d'évoquer les systèmes d'irrigation, qui supposent le détournement des cours d'eaux, la stabilisation de leurs courants par des bassins de rétention et des écluses, le contrôle de leur direction et de leurs flux par des canaux et par des vannes. À une échelle moindre, des agencements de tuyauteries, alimen-

121. (Bertalanffy [1968] 2015).
122. Voir plus haut, section 2.5.

tés par des citernes, construisent des mécanismes hydrauliques complexes, attestés dès l'Antiquité dans les fontaines à programme[123]. Les horloges à eau, puis mécaniques à partir de la fin du XIII^e siècle, exécutent une procédure simple : compter, de manière régulière, et incrémenter à partir de là divers registres (heures, jours, semaines, saisons, phases de la lune, etc.) selon des règles plus ou moins complexes. On peut les compliquer en les agrémentant de divers mouvements de figures automates. La machine naturelle ici est le simple écoulement régulier de l'eau, ou un poids dont l'accélération est régularisée par un dispositif d'échappement. Les montres ont fourni le mécanisme des premières machines à calculer, et le lien profond qui rapproche montres et ordinateurs a déjà été établi[124].

Toutes ces machines (le système d'irrigation, l'horloge à eau ou à poids) sont entièrement centrées sur la transmission d'un flux ou d'une énergie (eau, gravité) réguliers. L'origine du flux (les sources, l'attraction terrestre) y est naturelle, et il n'y a pas vraiment de « machine-outil » à l'extremité du flux si ce n'est, pour une horloge, les aiguilles du cadran. Qu'on puisse reconnaître dans un objet dynamique de l'expérience un *mécanisme de transmission* semble ainsi suffisant à ce qu'on puisse y décrire et implémenter des procédures. C'est cette idée que nous allons développer à la section suivante.

11.4 L'ordinateur comme contrôle de transmission

Dans cette section, nous achevons l'élucidation du rapport entre machines et programmation en nous intéressant à une classe très spécifique d'entre elles, les contrôles de transmission, qui sont le plus souvent des mécanismes internes à d'autres machines. Notre argument est que l'ordinateur, tel que nous le connaissons aujourd'hui, est essentiellement une telle machine. Nous commençons par présenter cette notion, que nous avons trouvée chez Marx, et l'importance qu'elle revêt dans son analyse de la puissance transformatrice des machines industrielles (§ 205). Nous montrons ensuite comment un ordinateur pourrait être construit à partir d'une transmission élémentaire, celle d'un système d'irrigation, à l'aide du modèle des *circuits booléens* (§ 206). Cela nous permet d'analyser en quel sens on peut dire que l'ordinateur est une *machine universelle* (§ 207). Une première conséquence de cette propriété est sa capacité à transformer le monde de la production

123. (Usher et Dupont 1988, p. 131-133).
124. (Rohrhuber 2018).

industrielle, notamment donnant à la machine une flexibilité totale (§ 208). Une seconde conséquence est sa capacité à automatiser les ratiocinations elles-mêmes et donc à transformer l'investigation elle-même (§ 209) – thème central que nous reprenons dans toute la dernière section de ce chapitre.

§ 205. Transmission et mécanisme

Il est possible de faire correspondre notre définition très générale de *machine* à celle que Marx donne de la machine industrielle, en trois composantes[125] :

1. La source d'énergie stable correspond à l'idée que la machine est un objet dynamique, animé d'un mouvement propre, c'est-à-dire d'un automatisme. Cela signifie, en termes logiques, qu'il y a une inférence par défaut que l'objet, d'un moment à l'autre, aura changé selon un schéma répétitif (par exemple un mouvement linéaire uniforme), du fait de la stabilité de la source.
2. La machine-outil correspond à l'idée d'une interaction réglée avec l'environnement. C'est là que se situent les opérations auxquelles peuvent se référer des significations : pousser, tirer, couper, brûler, etc. On se rappelle que Simon disait de la machine qu'elle devait être d'abord conçue à ses interfaces, c'est-à-dire dans la loi de son interaction avec son environnement externe. On notera que nous avons spontanément cité des verbes d'action pour décrire cette interaction. On conçoit en effet d'abord la machine comme modifiant son environnement. Cependant, la machine peut également *réagir* à des phénomènes externes, comme le métier à tisser de Jacquard adapte les opérations de tissage aux cartes perforées qui défilent.
3. La dernière composante de la machine identifiée par Marx, la transmission qui modifie le mouvement généré par la source d'énergie afin de l'amener aux machines-outils qu'elle commande, est sans doute la moins évidente à percevoir dans notre définition. Or c'est celle-ci qui nous intéresse. Elle correspond au principe de fonctionnement interne de la machine, qui modifie une énergie ou un mouvement brut et indifférencié en des énergies ou mouvements particuliers, requis par les machines-outils.

Ici *mécanisme* doit être pris dans un sens restreint, qui désigne tout dispositif servant à transmettre de manière réglée le mouvement spatial (bien qu'il soit également le principe de fonctionnement de la machine, son mécanisme au sens

125. (Marx [1867] 2006, p. 418).

large)[126]. Les mécanismes dans ce sens restreint sont étudiés par la technique qui s'appelle *mécanique* (qu'il faut donc distinguer de la science du même nom). Par exemple, la préface d'un ouvrage sur ce sujet les définit ainsi :

> Parmi les éléments des machines, un grand nombre n'ont d'autre objet que de transformer le mouvement produit par une source d'énergie de manière à la rendre utilisable pour certaines fins déterminées. Les éléments qui concourent à la transformation du mouvement constituent un « mécanisme ».
>
> Le mécanisme apparaît dès lors comme une série de pièces qui se transmettent un mouvement en le transformant progressivement. De cette définition résulte qu'un élément constitutif de mécanisme ne peut se concevoir employé isolément, mais toujours en connexion avec l'élément qui le précède ou le suit[127].

Dans l'analyse de Marx, le mécanisme de transmission se trouve être au cœur de la révolution industrielle, bien plus que la substitution des sources d'énergie humaine et animale par la machine à vapeur. L'outil manié par la personne humaine se transforme radicalement lorsqu'il devient une machine-outil mue par un mécanisme de transmission, car il n'est plus contraint par la physiologie humaine et peut être adapté aux besoins exacts et précis du processus de production[128], et c'est ce mécanisme qui, en retour, requiert le flux énergétique stable que donne la machine à vapeur. C'est seulement par la précision de la machine-outil et par l'exactitude de la transmission que la machine se débarrasse définitivement de l'approximation artisanale que Koyré jugeait pré-moderne, mais qui est surtout pré-industrielle.

Il y a cependant bien plus que cela, car la transmission permet également de passer de l'idée d'une simple coordination de machines-outils à celle d'un *système de machines*, qui ne se substitue pas à l'ouvrier individuel, mais à l'atelier tout entier. Un tel système réorganise le processus de travail selon sa logique propre, ajustant exactement les mouvements des différentes machines-outils de manière à atteindre une efficacité maximale. Il y a là un changement de paradigme que Marx signifie par l'évocation d'une métaphore célèbre et grandiose :

> C'est comme système articulé de machines de travail qui ne reçoivent leur mouvement que d'un automate central par l'entremise de la machinerie de transmission que l'exploitation mécanisée a sa configuration la plus développée. La machine isolée y a fait place à un monstre mécanique dont le corps emplit des corps de bâtiment entiers de la fabrique, et dont la force démoniaque, un temps dissimulée par le mou-

126. Voir plus haut, § 204.
127. (Vander Haeghen [1957] 2015, p. 7).
128. (Marx [1867] 2006, p. 419).

> vement précis et presque solennel de ses gigantesques membres, éclate dans la folle et fébrile sarabande de ses innombrables organes de travail proprement dits[129].

Or nous avons déjà noté la proximité conceptuelle de la réorganisation industrielle du processus de production et de la programmation informatique[130]. Si c'est bien le mécanisme de transmission qui joue un rôle central dans cette réorganisation, c'est à son étude qu'il faut à présent nous attacher.

Intuitivement, on voit bien comment une machine de Turing ou un système de Post pourraient *servir* comme des transmissions complexes de mouvements ou de flux. Il suffit que chaque symbole y représente un mouvement possible d'une machine réelle. Dans le cas d'une machine de Turing, la transformation d'une séquence de symboles inscrite initialement sur le ruban en une autre inscrite à sa terminaison, jouerait alors le rôle d'une modification de mouvements donnés en entrée, en mouvements à actionner en sortie. La machine de Turing jouerait donc le rôle d'un mécanisme de commande de la machine. Nous y revenons tout à l'heure[131].

Cependant, pour l'instant, nous devons établir la proposition inverse, à savoir s'il est possible de *réaliser* un ordinateur dans un mécanisme de transmission, autrement dit d'y programmer une procédure mathématique quelconque. Si tel était le cas, nous aurions établi comment il est possible que les mécanismes de transmission – ces phénomènes très concrets et très simples de l'expérience – puissent jouer le rôle de l'exécutant dont s'occupe la programmation, cette pragmatique mathématique, telle que nous l'avons caractérisée dans les deux premières sections de ce chapitre.

Nous cherchons donc à comprendre pourquoi des raisonnements mathématiques, c'est-à-dire portant sur les propriétés des nombres, les relations entre ensembles, les espaces géométriques, etc. peuvent en droit être convertis en des processus concrets réalisables par des machines physiques.

Comme nous l'avons vu, les procédures et les raisonnements mathématiques, aussi longs et complexes soient-ils, sont toujours finis ; mieux que cela, ils sont exprimées dans des théories définies exactement (en général arborant plusieurs schémas d'induction), donc constituées d'un nombre fini de symboles et de règles élémentaires[132]. Il est donc possible de représenter ces symboles et ces règles par

129. (Marx [1867] 2006, p. 428).
130. Voir plus haut, section 5.3.
131. Voir plus loin, § 217.
132. Voir plus haut, § 194.

des entiers naturels, grâce à ce qu'on appelle l'arithmétisation, c'est-à-dire des mécanismes de codage *qui préservent les règles*. Cela veut dire que les formations de formules à partir des symboles, et les dérivations de formules les unes des autres peuvent être représentées par des calculs sur les entiers naturels[133]. Cela implique que tout espace systématique de significations, qu'il soit fini ou génératif, peut être représenté par le système génératif le plus simple qu'est $\mathbb{N}$.

Par ailleurs, la thèse de Church-Turing avance que toutes les formules de calcul sur les entiers naturels peuvent être représentées par des machines de Turing. Ce que nous devons donc montrer, c'est la possibilité de réaliser une machine de Turing dans un mécanisme de transmission. C'est l'objet du développement suivant.

§ 206. Construction d'un circuit booléen

Malgré sa simplicité conceptuelle, construire une machine de Turing n'est pas si aisé que cela. La plupart des constructions disponibles utilisent des composants électroniques avancés – qui supposent donc ce qu'on veut ici construire. Une construction purement mécanique, réalisée en LEGO par le projet RubENS à Lyon en 2013, a requis plus de 6000 briques. Il ne s'agit donc pas d'une construction simple[134]. Or, pour notre réflexion, il est important de nous représenter le plus simplement possible ce que nous entendons par la projection d'une procédure sur un mécanisme.

Une telle possibilité se voit mieux à partir d'une autre machine formelle, où l'idée de transmission apparaît de manière plus transparente. Il s'agit des circuits booléens, qui se présentent comme un graphe dirigé connectant des portes logiques, traditionnellement les portes de la conjonction ET, et de la négation NON. Il est démontré[135] que chaque machine de Turing peut être représentée par un circuit booléen d'une taille indéfinie (comme le ruban de la machine de Turing). De manière plus concrète, Petzold (2000) montre comment construire un ordinateur de von Neumann à partir de seules portes logiques.

Afin de construire un circuit booléen, seul est nécessaire un générateur de flux, qu'il s'agisse de lumière, d'électricité, d'eau, ou d'une force quelconque, qui se répand librement dans tous les conduits qui lui sont accessibles, un fil électrique, un conducteur de lumière, une canalisation, etc. Bien que la transmission élec-

133. (Boolos, Burgess et Jeffrey 2007, p. 187).
134. Voir le site du projet.
135. (Sipser 1997, p. 321-329).

trique soit naturellement notre modèle, nous illustrons, par continuité avec nos exemples précédents, et pour montrer également que ce mécanisme n'a rien à voir avec la nature matérielle de ses composants, la construction d'un circuit booléen par la modification de transmissions hydrauliques. Elles véhiculent mieux, selon nous, l'intuition qu'une action dans l'expérience se trouve au fondement de toute programmation.

Il faut d'abord supposer que, de la source d'eau originelle, et d'ailleurs de tout point du réseau hydraulique, un nombre indéfini de canaux peuvent être dérivés – la force du flux s'ajustant à ce paramètre pour rester au-dessus d'une valeur minimale. Cela est moins aisé à concevoir dans le cas d'un circuit hydraulique qu'électrique, mais on peut imaginer que la section du flux est minimale par rapport à la source d'eau disponible, qui est stockée à des endroits stratégiques du réseau par des bassins de rétention. Par ailleurs, chaque canal peut être doté de barrières (d'interrupteurs dans le cas de circuits électriques) qui peuvent être actionnés soit manuellement, soit par un flux hydraulique acheminé en ce point par un autre canal. Il faut enfin imaginer qu'on peut construire des réseaux d'une complexité indéfinie, les canaux ne se croisant que si le croisement est explicitement spécifié, par exemple en construisant le réseau en hauteur, grâce à un système d'écluses ou de pompes.

Imaginons donc une machine simple, initialement constituée d'un flux hydraulique s'écoulant régulièrement de la source jusqu'à un fleuve par une seule canalisation. Programmer cette machine consiste à dériver des canalisations supplémentaires et y placer des barrières manuelles ou automatiques. La mise en opération de ce réseau (l'exécution du programme) consiste à activer une ou plusieurs barrières manuelles sur des canaux reliés directement à la source, qui figurent les entrées binaires du circuit et, une fois les flux stabilisés, à relever la présence ou l'absence d'eau en des points situés en aval, qui en figurent les sorties.

Ce circuit peut être construit à l'aide de deux opérations. La première est de creuser un canal à partir d'un point existant du réseau vers un nouveau point. La seconde est de construire une barrière obturant un canal existant, selon que celle-ci est actionnée par un flux provenant d'un point tierce. Un mécanisme classique pour réaliser cette deuxième action est l'assemblage *piston - crémaillère - pignon - arbre* : le flux actionne un piston qui met en mouvement rectiligne une crémaillère liée à un pignon qui transforme ce mouvement en mouvement circulaire. Celui-ci fait tourner un arbre sur son axe, levant ou abaissant la barrière.

On peut réduire ces deux opérations à une seule. Chaque canal qui est creusé entre deux points est toujours équipé d'une barrière, que celle-ci soit alimentée par un flux en amont ou non. Cette opération unique peut être représentée par la fonction suivante, dans le style constructif de Martin-Löf :

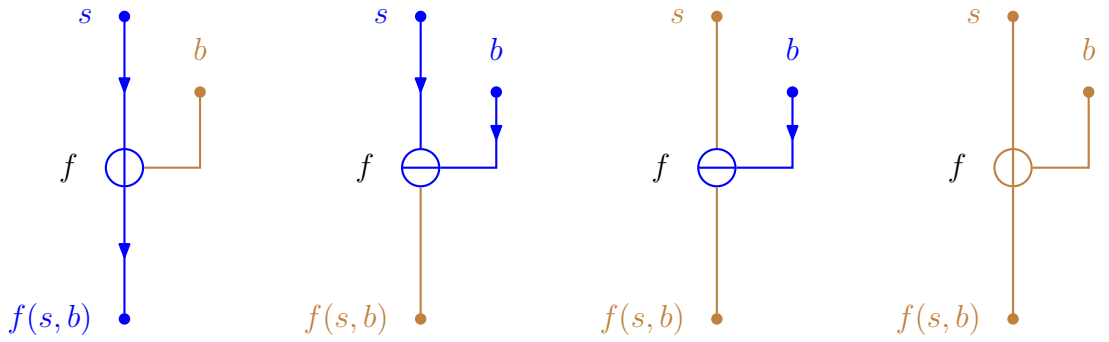
$$\frac{s : \text{Point} \quad b : \text{Point}}{f(s,b) : \text{Point}}$$

où s représente un point source d'où part le nouveau canal et b un point qui alimente la barrière de ce dernier. Il est entendu que b peut être un point sec T, auquel cas la barrière n'est jamais actionnée. $f(s,b)$ désigne aussi bien le point situé en aval du canal nouvellement construit que l'acte de construire le canal. Il existe une fonction d'évaluation e s'appliquant à tout point, qui peut prendre une valeur binaire V ou F, selon que la source s et la barrière b prennent les valeurs suivantes :

$e(s)$	$e(b)$	$e(f(s,b))$
V	V	F
V	F	V
F	V	F
F	F	F

La fonction f se visualise aisément dans la figure 11.1, où la couleur bleue représente le flux de l'eau, et l'ocre son absence.

FIGURE 11.1 – La fonction f, et ses quatre configurations

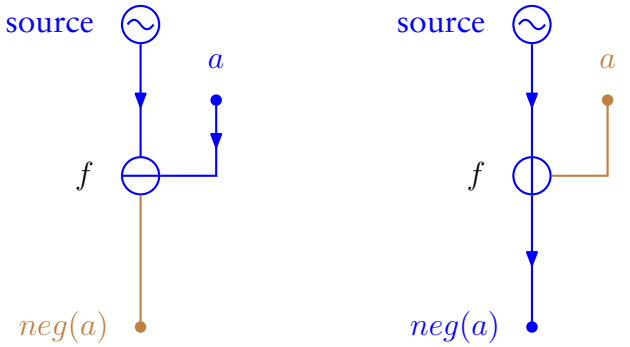


On peut construire la porte de la négation avec le circuit suivant, où source représente la source originelle, toujours alimentée :

$$neg(a) = f(S,a)$$

Cette expression signifie qu'à partir du point a un canal est creusé vers une barrière qui peut bloquer un canal creusé de la source S vers le point $neg(a)$ qui donne le résultat escompté, comme on peut le voir sur la figure 11.2.

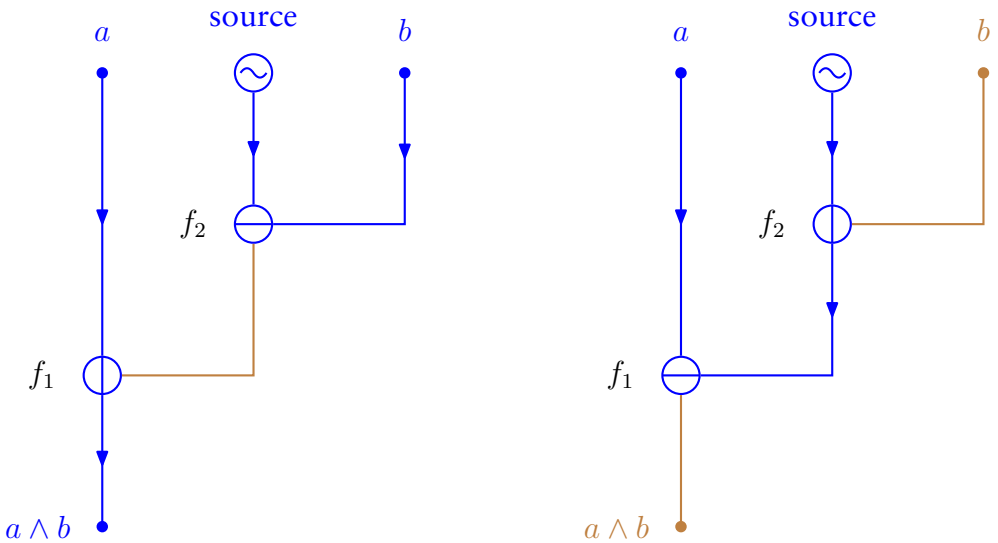
FIGURE 11.2 – La fonction $neg(a)$, pour $e(a) =$ vrai, et $e(a) =$ faux



Enfin, une porte de conjonction ET est construite grâce à deux fonctions f emboîtées, ainsi qu'on peut le visualiser dans la figure 11.3.

$$and(a, b) = f(a, f(S, b))$$

FIGURE 11.3 – La fonction $a \wedge b$, illustrée pour $e(a), e(b) =$ vrai, et pour $e(a) =$ vrai, $e(b) =$ faux



On peut de la même manière construire toutes les autres portes logiques. Tous ces actes de construction, qui partent de l'opération f, correspondent à des actes concrets, de creuser des canaux, poser des barrières, etc. et leur évaluation en V et F à la libération de l'eau dans les canaux amont du circuit. Il y a ici une correspondance élémentaire entre ces opérations et la programmation d'un circuit booléen.

En d'autres termes, nous montrons ici la possibilité de modifier une transmission hydraulique simple à partir d'un seul dispositif élémentaire, de manière à pouvoir y représenter tout circuit booléen, et donc potentiellement n'importe quelle procédure effective. Cette opération est si simple qu'elle apparaît en droit avec l'irrigation, même si ses possibilités indéfinies de complication ne commencent à être explorées qu'avec l'apparition des clepsydres, et beaucoup plus tard, avec celle des horloges mécaniques. La construction de circuits électriques se fonde également sur ce même acte : dériver un câble à partir d'un circuit, et placer un interrupteur alimenté électriquement sur ce câble.

Nous n'affirmons nullement que ces actes sont essentiels, ni que les circuits booléens représentent une forme élémentaire de calculabilité. Il est tout à fait possible qu'il existe des objets de l'expérience descriptibles par d'autres systèmes de significations, ayant la puissance des machines de Turing – par exemple des automates cellulaires, des réseaux de neurones, ou des calculateurs quantiques[136]. L'important pour notre propos n'est pas d'identifier le « bon » système de significations, mais d'en montrer un qui soit réalisable par des gestes connus depuis l'Antiquité et qui ait la puissance des machines universelles.

§ 207. La machine universelle

La démonstration précédente nous permet donc de soutenir l'hypothèse que le programmeur, tel que nous le connaissons aujourd'hui, est un ingénieur qui conçoit des mécanismes de transmission. On peut prendre cette présentation de manière assez littérale. La programmation de l'ENIAC, considéré comme le premier ordinateur, se faisait par un réel agencement de câbles[137]. Il n'est pas non plus absolument anecdotique historiquement que l'informatique ait vu de nombreuses de ses découvertes fondamentales hébergées par les Bell Labs, l'organisation de recherche de l'opérateur de télécommunications américain AT&T, comme la théorie

136. Voir plus loin, § 214.
137. (De Mol 2016).

de la communication de Claude Shannon, le transistor, le système d'exploitation Unix ou le langage C. Enfin, depuis le développement d'Internet, les deux industries ont a nouveau largement convergé[138]. Pour un ordinateur, « transmettre » et « traiter » l'information vont de pair.

Comment distinguer le programmeur de tous les autres ingénieurs travaillant sur des transmissions ? Car l'ingénieur télécom, l'électricien, le mécanicien, l'ingénieur hydraulicien, etc. y travaillent également. On pourrait être tenté de répondre que le programmeur ne cherche qu'à modifier le comportement d'une machine, et non à la construire. Cependant une telle distinction entre *modification* et *construction* ne convient pas. D'un point de vue factuel, nous avons très tôt vu que la conception contemporaines d'ordinateurs – et en particulier de processeurs – avait été réduite à un travail de programmation[139]. Plus généralement, la conception de machines en général se fait aujourd'hui par CAO*, des logiciels complexes dont la frontière avec la programmation est ténue. Nous y revenons tout à l'heure.

La réponse intuitive, bien sûr, est que le programmeur travaille sur des transmissions bien plus complexes que les autres ingénieurs, des transmissions qui ont une mémoire, qui peuvent suivre des branchements conditionnels ou boucler en partie sur elles-mêmes. Cependant, comment traduire cette complexité en termes exacts ?

On peut faire appel ici à la théorie mathématique des automates (une branche de la théorie de la calculabilité) qui distingue plusieurs classes de machines selon leur capacité à reconnaître des configurations plus ou moins complexes de signaux. Par convention, on décrit les configurations possibles de signaux comme l'ensemble des suites finies (mais de longueur indéterminée) de 0 ou de 1. Les plus simples automates (dits *finis*, *finite automata*) traitent les signaux les uns après les autres. Fondamentalement, ils savent reconnaître des répétitions d'une configuration quelconque finie de 0 et de 1, qu'on peut appeler a, comme $\{(a^n | n \in \mathbb{N}\}$. Ils ne peuvent cependant reconnaître des familles de configurations plus complexes, comme $\{a^n b^n | n \in \mathbb{N}\}$, où a et b sont deux configurations finies de signaux[140]. Nous appelons dorénavant *mots* (*words*) ces configurations finies et fixées de signaux afin de les distinguer des configurations qu'il s'agit de reconnaître.

138. Voir plus haut, § 26.
139. Voir plus haut, § 26.
140. Dans cette présentation simplifiée de la théorie des automates, nous utilisons le *lemme du pompage* (Sipser 1997, p. 77-83, 115-119).

Pour cela des automates qui sont capables de mémoriser une quantité indéterminée de signaux sont nécessaires, dont un des modèles les plus simple est l'automate à mémoire empilée (*push-down automaton*). Il est appelé tel parce que chaque nouvelle mémorisation est placée « au-dessus de la pile » et qu'un appel à la mémoire la rétablit donc d'abord. Une telle machine peut résoudre le problème précédent, car elle peut *mémoriser* chaque occurrence de a dans sa pile, avant de les effacer une à une chaque fois qu'elle rencontre un b. Le signal a donc la forme requise si sa pile est bien vide à la fin de la réception des signaux.

Une telle machine a cependant encore des limitations. Si a, b et c sont trois mots, alors elle ne peut reconnaître la famille de configurations $\{a^n b^n c^n | n \in \mathbb{N}\}$. Pour cela deux piles seraient requises, mais on voit bien qu'une telle limite pourrait être indéfiniment étendue à des configurations toujours plus complexes.

Afin de résoudre ce type de problèmes dans le cas général, des machines de Turing sont nécessaires. Là, les signaux sont inscrits sur un ruban sur lequel la machine peut se déplacer librement et qu'elle peut modifier. Une telle machine peut, en se déplaçant de manière adéquate à chaque itération, barrer à la fois une occurrence de a, b et c et ainsi décider, quand elle ne trouve plus l'un de ces mots, si leurs répétitions sont en nombre égal. Il est évident qu'une telle procédure fonctionne quel que soit le nombre $a_1, a_2, \ldots, a_n$ de configurations dont il faut juger l'égalité de répétition. Un *automate linéairement borné* (*linear bounded automaton*) a un ruban d'une taille exactement égale au signal qu'il doit traiter. Une machine de Turing en général a un ruban d'une taille indéfinie (aussi grand que nécessaire à la réalisation du calcul), et peut reconnaître des configurations de signaux d'une complexité indéfinie, ou en tous cas, toute suite de signaux que nous serions disposés à appeler encore *configuration*, y repérant intuitivement une régularité que nous saurions décrire : il s'agit là de la thèse de Church-Turing.

Il existe une autre manière de présenter ces différentes classes de machines, qui nous semble plus intéressante pour notre propos car elle ne la caractérise pas en faisant appel à la notion de configuration de signaux, mais uniquement à la notion de *machine* elle-même. Elle découle d'un résultat peu connu de la théorie des automates, exposé pour la première fois par Boucher (1972) et repris par Jürgensen (2012).

Nous avions déjà relevé que la spécificité de la notion de *machine*, parmi les autres objets géométriques, était sa récursivité : la construction d'un objet étant un mouvement, et la machine ayant pour propriété le mouvement, elle était capable

(en droit) de construire d'autres machines[141]. Puisqu'on parle ici de mécanismes de transmission, la question est la suivante : peut-on construire un mécanisme de sorte qu'il reproduise fidèlement le comportement d'autres mécanismes, et jusqu'où peut-on aller dans cette polyvalence ? Peut-on construire un mécanisme universel, qui permettrait de se dispenser de construire tout autre mécanisme ?

Un mécanisme polyvalent, qu'on peut appeler $\mathcal{U}$, se comporterait de la manière suivante : il interpréterait la première partie du signal comme lui donnant la description du mécanisme à reproduire, qu'on peut appeler $\mathcal{A}$, et la seconde partie comme le signal s à traiter par ce mécanisme (éventuellement lui-même transformé selon une codification fixe). Claude Boucher appelle ϕ_1 la fonction de description qui transforme en signal de $\mathcal{U}$ le mécanisme cible $\mathcal{A}$, et ϕ_2 la transformation du signal de $\mathcal{A}$ en un signal de $\mathcal{U}$. On a donc :

(1) $\quad \forall s, \mathcal{A}, \mathcal{U}(\phi_1(\mathcal{A})\phi_2(s)) = \mathcal{A}(s)$

Les résultats de C. Boucher sont (entre autres) les suivants :

1. On peut construire un automate fini reproduisant tout ensemble fini d'automates finis. Un tel résultat est assez intuitif : du moment qu'on considère un groupe de machines répondant chacune de manière déterminée à un nombre fini de paramètres, on peut construire une machine incluant tous ces paramètres, ainsi qu'un paramètre de plus indiquant quelle machine simuler.
2. Il n'existe pas d'automate fini reproduisant *tous* les automates finis, mais il existe une machine à mémoire empilée qui en est capable. Là encore, ce résultat est intuitif, puisque les automates finis peuvent reconnaître des répétitions de mots a indéfiniment complexes. Il faut donc une machine capable de mémoriser n'importe quel mot avant d'en tester la répétition.
3. Il existe de la même manière une machine à mémoire empilée qui reproduit tout ensemble fini de machines de cette même classe. Mais il n'existe pas, pareillement de machine à mémoire empilée universelle. Il faut pour cela un automate linéairement borné.
4. Le même résultat s'applique pareillement aux automates linéairement bornés[142].
5. Enfin, nous rappelons ici le résultat bien connu, puisqu'il est démontré par Turing lui-même, qu'il existe une machine de Turing universelle pour toutes les classes de machines, y compris les machines de Turing.

141. Voir plus haut, § 119.
142. Voir là-dessus Jürgensen (2012, p. 156).

Autrement dit, la classe des machines de Turing est la classe minimale qui permet de concevoir une machine universelle dans sa propre classe. Lorsqu'on cherche à concevoir une machine universelle pour les automates finis, on est amené à concevoir une machine qui a les mêmes propriétés qu'une machine à mémoire empilée, et une fois qu'on dispose d'un tel concept, chercher à concevoir une machine universelle pour toutes ces nouvelles machines amène à concevoir un automate linéairement borné, etc. Les machines de Turing semblent être la clôture minimale des classes des machines selon la relation de simulation universelle. Un tel résultat nous semble (mais il s'agit là d'une conjecture) apporter une confirmation complémentaire à la thèse de Church-Turing – que cette clôture minimale est également maximale, et qu'il n'existe pas de classe de machines plus puissantes que celles de Turing.

Un tel résultat permet donc de présenter la programmation comme la conception de mécanismes de transmission dans toute leur généralité possible, *ce qui équivaut* au fait qu'elle peut se réaliser sous la forme de signaux envoyés à un seul mécanisme de transmission universel, qui prend la place de tous les autres. C'est *de facto* ce qui se réalise à partir des années 1990, lorsque la miniaturisation et le coût de transistors baissa suffisamment pour que le contrôle de transmission interne à la plupart des machines industrielles soit dévolu à des processeurs appelés micro-contrôleurs[*143]. Cette remarque pointe vers la forme la plus élémentaire d'effectivité qu'a la programmation : celle tout simplement d'augmenter l'effectivité des machines industrielles en général. Avant de développer cette idée, il nous faut cependant revenir sur le terme d'*universelle* par lequel nous avons qualifié la machine $\mathcal{U}$.

Dans un ouvrage de réflexion sur sa discipline, l'informaticien Edward A. Lee s'élève contre les confusions que cette expression entraîne, car elle laisse supposer qu'un ordinateur peut simuler n'importe quelle autre machine ; pour lui, $\mathcal{U}$ est universelle relativement à la classe des machines de Turing, qui sont des machines très spéciales[144]. Un ordinateur numérique, non seulement ne peut pas laver le linge, mais il est même des fonctions mathématiques qu'il ne peut calculer et que d'autres machines calculent très bien. Lee prend ici l'exemple du nombre π, qu'un ordinateur ne peut qu'approcher, mais qu'une machine « analogique » peut très bien réaliser dans un dispositif matériel, par exemple un ballon qu'on gonfle. Enfin,

143. (Trevennor 2012).
144. (Lee 2017, p. 210).

Lee remarque que plusieurs ordinateurs en interaction concurrente ne peuvent en général être simulés par une machine universelle : on en a vu les raisons avec l'exemple des calculs distribués[145].

Lee a tout à fait raison sur ces points, cependant la portée qu'il attribue au terme d'*universalité* est trop restreint. Ce n'est pas seulement d'une classe de machines que $\mathcal{U}$ est la simulation universelle, mais surtout de *toutes les procédures effectives* qu'il est possible de concevoir (en admettant la thèse de Church - Turing). Cela implique, en particulier, que *tous les dispositifs de commande* de machines particulières peuvent être simulés par $\mathcal{U}$. Ce qui se passe à l'intérieur de mon lave-linge peut être chaotique, cependant je peux *régler* certains paramètres pertinents de ce chaos (vitesse, température, essorage, etc.) afin d'obtenir des résultats différenciés.

Par ailleurs, les raisons pour lesquelles je peut contrôler ce chaos local sont elles-mêmes procédurales, et donc représentables par $\mathcal{U}$. Je sais par exemple que le linge serait projeté contre les parois du tambour par la force centrifuge, si celui-ci n'était équipé de redans (des petites lames saillantes) ramenant le linge vers l'intérieur. C'est ce mécanisme qui importe pour que le linge soit nettoyé, et un tel mécanisme, rappelons-le, n'est qu'un ensemble de règles d'inférences que je m'autorise à faire concernant ce phénomène de l'expérience[146].

Quant à l'exemple de la machine à calculer π, il n'est pas question de nier qu'il y a de nombreux phénomènes qui ne peuvent pas être reproduits de manière procédurale, comme un mouvement circulaire si on ne sait que tracer des segments – c'est bien d'ailleurs, pour cela, qu'Euclide *postule* la capacité de réaliser un tel mouvement. $\mathcal{U}$ n'est pas universelle relativement à tous les mouvements et à tous les gestes qu'il serait possible d'effectuer, mais à leur *composition*.

Enfin, Lee a raison de pointer que des machines en interaction peuvent ne pas pouvoir décrites comme une machine *unique*. C'est justement pour cela qu'on commence alors à parler de *système*. Nous revenons sur ce point au chapitre suivant.

Il est un dernier point à faire concernant cette notion de *machine universelle* : nous l'avons introduite, et montré sa possibilité matérielle, par le biais des mécanismes de contrôle de transmission. Il ne faut pas en conclure que cette possibilité est exclusive. Ce qui importe en général est d'avoir un phénomène dont les

145. Voir plus haut, § 52.
146. Nous revenons en longueur sur ce point à la section suivante 11.5.

règles d'interaction sont assez riches pour que les opérations sur les entiers naturels puissent y être représentées et composées entre elles ; d'autres phénomènes physiques que les transmissions de flux sont envisageables à cet égard. Nous revenons sur ce point à la section suivante.

§ 208. La résorption de la machine-outil

Les développements précédents montrent combien la programmation se situe naturellement dans la continuité de la révolution industrielle, telle que la décrivirent Babbage et Marx. Babbage avait déjà montré que l'automatisation ne devait pas être vue seulement sous l'angle des gains de productivité, mais également sous celui des nouvelles opérations qu'elle rendait possible, notamment grâce à la coordination et à la précision des opérations qu'elle permettait[147]. Les ordinateurs, lorsqu'ils prennent le contrôle des machines industrielles, permettent leur coordination indéfinie, bien plus que ne pourrait le faire une simple transmission mécanique. Ainsi, par exemple, les robots dans un entrepôt logistique entièrement automatisé savent aujourd'hui optimiser leurs trajets et leurs tâches, connaissant les plans des autres robots[148]. Nous allons ici développer le bénéfice de précision.

On se rappelle que, pour Marx, le mécanisme de transmission libère la flexibilité de la machine-outil, qui peut ainsi adapter exactement son geste au processus de production. Plus la transmission du mouvement est précise, plus la machine outil peut se simplifier pour aller à sa fonction essentielle. La découpe au laser, qui opère point par point, est infiniment plus précise qu'une découpe mécanique qui doit, selon l'opération à effectuer, utiliser des tours, des fraiseuses ou des perceuses. Pour préciser cette idée, nous développons l'exemple de la fabrication des verres ophtalmiques.

Un verre ophtalmique simple est caractérisé par sa courbure, qui lui permet de corriger une myopie ou une hypermétropie données. Cette courbure est obtenue traditionnellement en taillant la face arrière du verre avec un moule en métal qui épouse cette courbure, et qui a donc la forme d'un dôme plus ou moins aplati. Le verre est appliqué sur ce moule avec des mouvements de rotation qui fraisent sa matière, jusqu'à donner à sa face arrière la courbure du moule, en négatif. On appelle cette technique *surfaçage* (en bon français, cela correspond au *taillage* du verre). Cette technique a le défaut qu'elle peut être aussi précise que le nombre de

147. Voir plus haut, § 88.
148. Voir plus loin, § 229.

moules que possède l'atelier de fabrication. Étant donné que la correction s'exprime en quarts de dioptries sur une échelle allant en général de -8 (myopie) à +8 (hypermétropie), 64 moules sont en théorie requis.

Cela reste raisonnable, mais le devient beaucoup moins lorsqu'il s'agit de corriger la presbytie avec des verres progressifs qui corrigent aussi bien la vision de près que la vision de loin en adoptant une surface qui évolue progressivement d'une puissance à une autre, de la zone haute à la la zone basse du verre. Ils évitent d'avoir recours à des verres bi-focaux, peu esthétiques. Cependant, les verres progressifs sont complexes à tailler. Non seulement ils requièrent une combinaison de puissances, ce qui multiplie d'autant le nombre de moules requis, mais ils introduisent aussi des aberrations sur la vision latérale et d'autres désagréments. De ce fait, les verres progressifs sont depuis leur invention dans les années 1960 la source d'une recherche continue sur la vision presbyte pour corriger ces défauts. On a ainsi découvert comment pallier nombre de ces défauts en dessinant des géométries de courbure complexes et asymétriques, optimisés pour chaque zone de vision. Cependant, ces géométries ne peuvent pas être simplement reportées sur des moules standard, car elles dépendent d'autres paramètres de la vision, qui sont spécifiques à chaque patient. Avec la technique traditionnelle de surfaçage, il faudrait ainsi concevoir un moule pour chaque prescription !

Jusqu'au milieu des années 1990 la conception de verres progressifs était donc l'objet de compromis entre la qualité du verre et les contraintes industrielles. Cela a été révolutionné par l'invention du *surfaçage numérique*. Les machines numériques de taillage n'ont pas de moule, mais une pointe de diamant capable de fraiser le verre point par point. Cette pointe est associée à un bras robotique extrêmement précis, agissant au micromètre, sous le contrôle d'un programme lisant un fichier indiquant les coordonnées géométriques de la surface à tailler. Ce fichier est spécifique à chaque prescription et est généré par un calculateur en fonction des paramètres optiques mesurés sur le patient par l'ophtalmologue et par l'opticien. Aujourd'hui, jusqu'à vingt paramètres sont mesurés pour générer une géométrie adéquate aux besoins de chaque patient. Cette fonction de calcul concentre donc une part essentielle de la recherche des fabricants de verres ophtalmiques. La qualité de cette fonction de calcul détermine celle de la vision.

On voit bien ici comment le contrôle de la transmission du mouvement permet de produire des objets destinés à des besoins chaque fois uniques, d'une qualité inégalable par des moyens mécaniques traditionnels, et cependant selon des proces-

sus entièrement standardisés. La fabrication additive (ou « impression 3D » dans le langage courant) est l'aboutissement de la capacité à produire toute sorte d'objets à partir d'un processus unique de contrôle de transmission. Cette technique, qui se diffuse rapidement dans tous les secteurs industriels depuis les années 2000, est un processus contrôlé par ordinateur qui crée des objets tridimensionnels par dépôt de matériaux, généralement en couches, selon divers procédés (lasers pour les métaux et céramiques, polymérisation et extrusion pour les polymères, etc.) (voir figure 11.4). Elle est universelle au sens où l'objet à produire est entièrement défini par un fichier numérique donnant ses coordonnées stéréométriques, et qu'il peut donc prendre toute forme calculable. La fabrication additive permet ainsi la création de pièces complexes qui seraient trop difficiles ou trop coûteuses à produire à l'aide des techniques de fabrication traditionnelles (injection, usinage, etc. – les techniques traditionnelles de fabrication restent cependant en général préférables pour les grandes séries de production).

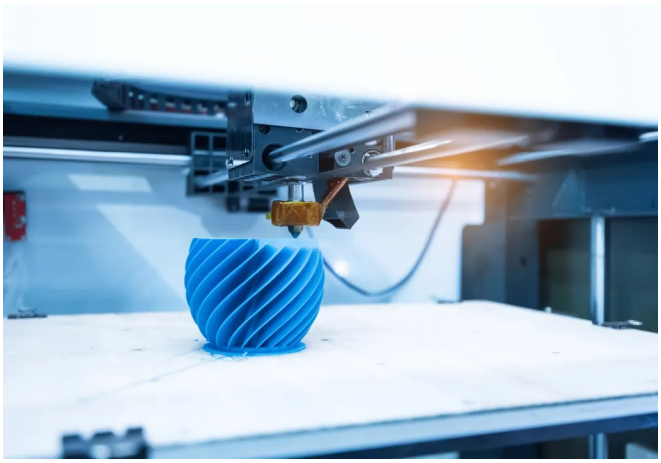


FIGURE 11.4 – Illustration de la fabrication additive
Source : www.autodesk.com/products/fusion-360

La fabrication additive montre donc la possibilité qu'a le mécanisme de transmission de « phagocyter » presque entièrement la machine-outil jusqu'à la réduire à une simple pointe laser. Il devient la machine tout entière, transformant une énergie stable et uniforme en une séquence indéfiniment précise de gestes élémentaires – déplacer la pointe dans l'espace et régler l'intensité du laser, pour simplifier.

Il s'agit là d'une première explication du constat de Haigh et Ceruzzi, que

l'informatique est une « technologie dissolvante »[149]. Elle termine de libérer la machine-outil de toute contrainte matérielle ou cinétique, de sorte que celle-ci s'identifie presque entièrement à l'opération qu'il s'agit de mener : couper, chauffer, injecter, etc. Toute la responsabilité de la forme, c'est-à-dire du mouvement, est transférée au mécanisme de transmission, lui-même entièrement contrôlé par un programme.

Nous sommes à présent en mesure de répondre aux trois objections que nous avions laissées en suspens après l'exposé de notre interprétation contractuelle de la programmation[150], à savoir qu'elle ne permet pas vraiment de mettre en avant la spécificité des ordinateurs relativement aux autres machines industrielles, qu'elle semble confondre programmation et construction de machines. et qu'elle ne fait pas droit à la nature mathématique de la programmation. Ces trois objections sont à présent résolues. La programmation peut être entendue en un sens large ou restreint. En un sens large, toute construction ou modification d'une machine afin de résoudre un problème peut en effet être nommée programmation : la convergence des machines industrielles et des ordinateurs, que nous avons tenté de rendre sensible ici, montre qu'il est de plus en plus délicat de séparer les deux activité. En un sens restreint, on peut réserver *programmation* à la seule modification de machines universelles. Pour ces machines, construire et programmer sont deux tâches identiques, car elles arborent de manière exemplaire la propriété centrale des machines, d'être des artefacts « récursifs » (capables de se produire eux-mêmes) ; et c'est de cette manière qu'on peut voir le caractère essentiellement mathématique de la programmation.

§ 209. La ratiocination

Cette section a permis de comprendre pourquoi les procédures systématiques peuvent être en droit transposées dans les langages descriptifs des machines. D'une part toute théorie peut être représentée dans le système inductif infini le plus simple, celui des entiers naturels, qui exhibe le schéma de la récursivité. D'autre part, ce système est lui-même représentable dans les systèmes de signification décrivant ces objets dynamiques de l'expérience que nous avons appelé *transmissions*, et qui désignent des écoulements réguliers et contrôlables de flux quelconques. Les transmissions sont les corrélats dans l'expérience de micro-modèles de systémati-

149. Voir plus haut, § 25.
150. Voir plus haut, § 60.

cité, c'est-à-dire de règles de comportement dont les conditions d'environnement sont entièrement explicitées et peuvent être parcourues exhaustivement. Une règle simple d'inférence – le flux va de A à B – peut être rendue indéfiniment plus complexe grâce à des opérations simples de diversion et de blocage, qui constituent son seul environnement.

Cette machine est un objet mathématique, car les significations par lesquelles on lq décrit sont de nature procédurale – ses seules significations matérielles d'intérêt étant la fonction f constructive de nouveaux segments du réseau et la fonction e évaluatrice permettant de relever la présence ou non du flux en un nœud du réseau. Mais plus que son caractère mathématique, c'est sa nature discrète (non continue) qui est importante, qui la rend exacte. D'une part les gestes qui les construisent (tracer un canal, activer ou désactiver une barrière, résumés ici dans la fonction f) ne demandent qu'une précision grossière pour être effectifs, et d'autre part la fonction e se contente de mesurer la présence ou l'absence du fluide, et non son intensité.

Ainsi on ne peut dire que la machine *exécute* la procédure qu'en un sens dérivé, ce que le sens commun n'aurait aucune difficulté à convenir. La machine est simplement agencée pour que nous puissions réaliser certaines inférences sur son comportement, dont nous avons par ailleurs établi l'équivalence avec d'autres procédures inférentielles conçues dans d'autres systèmes de signification. C'est cela que signifie pour une machine d'exécuter une procédure. On peut remarquer ici que l'*automatisation* est la réalisation de cet agencement, lorsque ces inférences portent sur l'observation d'actions humaines.

Ce que la machine automatise donc d'abord, c'est la ratiocination, que les investigateurs ont toujours dédaigné, voire abhorré[151]. Ce n'est pas l'automatisation en tant que telle qui fait l'économie de jugements intermédiaires dans l'investigation – tel était déjà l'objet de la ratiocination. Le gain propre de l'automatisation est de remplacer cette ratiocination humaine par un objet dynamique *configuré à l'avance* pour y correspondre ; il doit être évalué dans les termes déjà décrits de la productivité, de l'exactitude et de la nouveauté qu'il rend possibles[152].

Cette description est bien entendu conforme au bon sens ; son seul mérite est d'expliquer pourquoi on peut trouver, au sein même des activités de l'esprit, certaines d'entre elles qui se laissent si bien « industrialiser », dans le sens même où

151. Voir ci-dessus, p. 639 et notre remarque à la fin du développement § 178.
152. Voir plus haut, section 5.4.

cela a été fait de tant d'autres activités « manuelles ». Le point commun essentiel de toutes ces pratiques régulières, c'est qu'elles se prêtent à des descriptions procédurales, qui récapitulent les investigations passées et permettent de les remplacer par des ratiocinations, qui ne sont rien d'autres que des séquences réglées de gestes intentionnels.

Aussi serait-il réducteur de limiter l'accroissement d'effectivité apporté par les ordinateurs à l'investigation à la seule économie de ratiocination. Il est assez évident qu'ils permettent également de conduire de nouvelles formes d'investigation scientifique et de nouvelles questions, dont Humphreys ([2004] 2007) a, parmi les premiers, donné la mesure. Il est intéressant de remarquer à cet égard qu'il utilise le même critère générique que celui proposé par Caliste et Carnino pour caractériser l'industrie[153], à savoir que des changements quantitatifs dans une discipline donnée, au delà d'un certain effet de seuil, entraînent également son changement structurel. La puissance de la ratiocination ayant été ainsi démultipliée de plusieurs ordres de magnitude, les questions de recherche des sciences s'en trouvent également bouleversées[154]. Au fondement de cela se trouve, selon Humphreys,

> un point qui ne devrait pas être sujet à controverse, mais qui n'est pas suffisamment souligné en philosophie des sciences, qu'une grande partie du succès des sciences physiques modernes est due au calcul. Cette activité peut sembler peu intéressante, mais lorsqu'il s'agit de passer de la théorie à l'application, elle est presque indispensable. Peut-être à cause de son manque de prestige, mais aussi pour des raisons structurelles plus profondes, la philosophie des sciences du xx^{e} siècle n'a pas accordé au calcul scientifique le crédit qu'il méritait, au-delà de la philosophie de la connaissance. Cela est curieux, car derrière une grande partie de la science physique se cache le principe suivant : *c'est l'invention et le déploiement de mathématiques réalisables [*tractable*] qui sont à l'origine des progrès des sciences physiques*. Chaque fois qu'il y a une augmentation soudaine des mathématiques utilisables, il y a une augmentation soudaine et concomitante du progrès scientifique dans le domaine concerné[155].

Nous pensons pouvoir à présent expliquer les raisons de ce fait. De la même manière que la mécanisation n'est pas en effet une simple possibilité ouverte par l'industrialisation, mais l'un de ses moteurs principaux[156], l'informatisation des

153. Voir plus haut, § 86.
154. (Humphreys [2004] 2007, p. 52-53).
155. (ibid., p. 55).
156. Voir plus haut, p. 719.

procédures ne doit pas non plus être vue comme une simple possibilité ouverte par la pensée systématique, mais au contraire comme l'un de ses moteurs.

Avant de développer cette idée, il nous faut cependant étudier deux autres interprétations importantes aujourd'hui du rapport entre procédures et machines, ce qui va nous permettre d'approfondir notre analyse du concept de *mécanisme* et son rapport à celui de *procédure*.

11.5 Programmes et mécanismes

La capacité des mécanismes de transmission à exprimer tous les contenus mathématiques a certainement frappé les esprits. Leur simplicité, et l'indifférence de leur comportement à la nature du flux transmis (eau, électricité, forces, lumière, etc.) a encouragé la formation des concepts abstraits de *signal* et d'*information*. On peut alors formuler la thèse suivante : si les sciences parviennent « avec une si déraisonnable effectivité[157] » à exprimer les lois de la nature en termes mathématiques, n'est-ce pas parce que la nature est tout entière traitement d'information ? En d'autres termes, tous les mécanismes ne sont-ils pas, fondamentalement, des mécanismes de transmission ?

On peut identifier cette thèse au pan-computationnalisme, puisqu'ici, il est affirmé que tous les mécanismes matériels sont des calculs et que la matière, ce concept métaphysique obscur, peut être avantageusement par celui, parfaitement mathématisable, d'information. Nous devons d'autant plus prendre au sérieux cette thèse métaphysique, appelée *pan-computationalisme*, que nous soutenons apparemment des thèses très proches, quand nous affirmons que les mécanismes ne sont que les objets de descriptions procédurales (mathématiques) de phénomènes de l'expérience. Cette confusion serait dommageable, puisque notre travail se revendique d'une stricte neutralité métaphysique : il serait paradoxal que nous défendions une thèse aussi spéculative. Par ailleurs, – puisqu'elle tient que les lois de la nature sont homogènes à des programmes – cette position tend à confondre les figures du scientifique et du programmeur, que nous souhaitons distinguer. Enfin et surtout, il nous faut approfondir la notion de *mécanisme* esquissée dans les sections précédentes, et montrer son caractère interne et nécessaire à l'attitude systématique de résolution de problème. Si elle apparaît si nécessaire aussi bien à notre appréhension technique que scientifique du réel, c'est qu'elle représente un

157. (Wigner 1960).

idéal théorique de transparence de l'explication et du contrôle des phénomènes de l'expérience. En cela, le pan-computationalisme est une continuation possible, sous une autre forme, de la philosophie mécanique du Grand Siècle. En tant qu'il accorde cependant un privilège exorbitant aux mécanismes de transmission et à la notion d'information qui en est dérivée, il est arbitraire.

Nous commençons par exposer le pan-computationalisme dans sa proximité avec les thèses développées dans les chapitres précédents (§ 210). L'essentiel de cette section consiste ensuite à approfondir la notion de mécanisme en nous appuyant sur la réflexion du philosophe Gualtiero Piccinini, qui se réclame du *nouveau mécanisme*, une doctrine épistémologique contemporaine. Pour réfuter le pan-computationalisme, Piccinini entreprend de montrer que les mécanismes procéduraux ne sont qu'une forme très spécifique de mécanisme. Cette thèse contredit la thèse du pan-computationalisme mais aussi bien la nôtre. L'examen serré de ses analyses va donc nous permettre de distinguer notre position épistémique de ces deux doctrines.

Nous exposons en § 211 la thèse de Piccinini, qui met en évidence quatre différences entre un mécanisme, pris dans son sens le plus large et l'exécution d'une procédure, qui en est une forme très spécifique selon lui. Après un examen rapide des deux premières distinctions, nous portons notre attention sur la troisième, (§ 212) qui nous conduit à critiquer la notion centrale d'*esquisse de mécanisme*, et sur la quatrième qui porte sur le caractère non-intentionnel d'un mécanisme (§ 213).

Ces analyses nous permettent d'expliquer que le privilège des mécanismes de transmission est avant tout lié à leur *transparence* épistémique, c'est-à-dire à la possibilité d'établir une correspondance directe entre une inférence et une transmission élémentaire – un chemin simple qui va d'un point à un autre (§ 214).

Concernant notre emploi fréquent de l'adjectif *computationnel* dans les deux sections qui suivent, on peut se référer à notre note à ce sujet en introduction de la section 1.3.

§ 210. La thèse de Church et la thèse de Galilée

Le mouvement de notre réflexion, dans les quatre premières sections de ce chapitre, a été le suivant. Plutôt que tenter de montrer comment il était possible que des procédures – qui relèvent du domaine des *significations* – pussent être « réalisées » dans la matière, nous avons au contraire affirmé que les machines et

les mécanismes étaient eux-mêmes des représentations procédurales, et donc *déjà homogènes* à des procédures. Lorsqu'on transpose une signification mathématique dans un système de significations décrivant le comportement d'une machine, on se situe toujours dans le plan des significations. Ce n'est que lorsque la procédure mécanique est mise à exécution qu'on peut juger si les vastes réseaux d'inférences ainsi tissés sont conformes à l'expérience.

Une telle thèse ne prête-t-elle pas à confusion ? Si les mécanismes sont des représentations procédurales, cela ne revient-il pas à affirmer qu'ils sont essentiellement des calculs et des transmissions d'information ? et cela ne conduit-il pas à ce qu'on appelle aujourd'hui *pan-computationnalisme*, cette thèse qui affirme que l'univers est composé d'information et de calculs, plutôt que de matière et de mouvements ?

Le pan-computationalisme offre en effet une réponse subtile à la question classique de la « déraisonnable effectivité des mathématiques » en physique[158], qu'exprime très bien l'informaticien français Gilles Dowek dans un court article, en affirmant que « la thèse physique de Church implique la thèse de Galilée[159] ».

Dowek appelle *thèse de Galilée* l'idée que « les lois de la nature sont écrites en langage mathématique », ce qu'il traduit plus précisément dans l'assertion que toute relation régulière entre une mesure expérimentale et ses paramètres doit pouvoir être exprimée mathématiquement. Il appelle *thèse physique de Church*, en s'appuyant sur les travaux de Gandy déjà évoqués[160], l'assertion que, pour tout système physique avec lequel un protocole d'interaction peut être défini, la relation entre les données qu'on lui fournit et les résultats qu'on en extrait est une fonction calculable. Gandy rend cette thèse probable en montrant qu'un système qui respecterait des principes physiques minimaux – d'abord, que ses composants ou sous-composants signifiants aient une densité maximale (il y a des « atomes ») et ensuite, que leur portée causale soient bornée localement (ils ne peuvent influencer d'autres composants que jusqu'à une certaine distance) – est équivalent à une machine de Turing. Dowek interprète cela au prisme de la théorie de l'information :

> Robin Gandy (1980) a montré que la thèse de Church physique est une conséquence de trois hypothèses sur la nature : l'homogénéité de l'espace et du temps, la limitation de la vitesse de propagation de l'information, la limitation de la densité de l'information[161].

158. (Wigner 1960).
159. (Dowek 2012, p. 2).
160. Voir plus haut, § 203.
161. (Dowek 2012, p. 3).

L'implication de la thèse de Galilée par la thèse de Church découle simplement des définitions données ci-dessus : toute expérimentation est un système physique doté d'un protocole d'interaction ; si donc la relation entre la mesure expérimentale et ses paramètres est une fonction calculable, alors elle est également, de manière triviale, également mathématique.

Ainsi, si toute explication scientifique de l'expérience peut être en droit décrite en termes procéduraux (mathématiques), c'est que les mécanismes qu'elle étudie sont eux-mêmes, de manière ultime, des procédures, c'est-à-dire que, de quelque manière qu'on entre en interaction avec eux, leur comportement se révèle être descriptible par des procédures. Pour Dowek, c'est bien le second fait qui est fondamental. Il l'exprime très clairement :

> Un effet secondaire de cette explication de la thèse de Galilée est que les lois de la nature peuvent être décrites non seulement dans le langage des mathématiques, mais aussi dans un langage conçu pour exprimer des algorithmes : un langage de programmation. En particulier, […], le langage des équations différentielles peut être considéré comme un langage pour définir des algorithmes : un langage de programmation[162].

Dans un sens strictement épistémique, une telle thèse conviendrait très bien à Dewey (et à nous-même) : elle ne fait que réaffirmer la nature fondamentalement procédurale des notions mathématiques[163]. Cependant, Dowek insiste sur le fait qu'il faut entendre cette thèse de manière réaliste : « l'Univers et notre description de l'Univers ne sont pas indépendants […] Cette thèse doit être comprise comme une thèse sur la Nature, et non sur le langage[164] ».

Mais il ne suffit pas de suspendre cette dernière affirmation métaphysique pour que nous tombions d'accord avec Dowek. Les principes – mêmes minimaux – qu'il demande d'accorder concernant la densité maximale et la vitesse maximale de l'information ne nous semblent pas être à l'origine du caractère procédural de la connaissance, mais plutôt des reformulations de nos propres exigences concernant la nature des règles, notamment le principe de localité (qui correspond à l'idée d'une vitesse maximale) et le principe d'induction (qui correspond à l'idée d'une profondeur maximale de l'analyse)[165]. Nous allons à présent examiner la réfutation du pan-computationalisme proposée par les nouveaux mécanistes, ce qui va

162. (Dowek 2012, p. 3).
163. Voir plus haut, § 173.
164. (Dowek 2012, p. 4, 5).
165. Voir plus haut, § 160.

nous permettre également d'approfondir la notion de mécanisme.

§ 211. Proximités et différences entre calcul et mécanisme

Le nouveau mécanisme est une théorie normative de l'explication scientifique, apparue au début des années 2000. Comme nous l'avons déjà succinctement indiqué, elle soutient que, dans de nombreux domaines scientifiques, et notamment ceux des sciences de la vie auxquels elle s'est initialement attachée, l'explication prend la forme de la description de « mécanismes », ceux-ci étant définis comme « des entités et des activités organisées de sorte qu'elles soient productives de changements réguliers du début (ou conditions initiales) à la fin (ou conditions terminales) » [166].

Il existe de nombreuses interprétations philosophiques de cette idée, dont Levy et Bechtel (2013) donnent un aperçu. Une interprétation forte affirme que tous les phénomènes causaux doivent pouvoir être réduits à des mécanismes, jusqu'à parvenir aux « mécanismes de base » décrits par la physique fondamentale. On a donc affaire ici à une thèse ontologique, qui pourrait être comparée à la réflexion de Gandy et Sieg. L'interprétation majoritaire est cependant épistémologique, et affirme que la représentation des mécanismes sous-jacents aux phénomènes constitue la visée de l'explication scientifique, évitant ainsi, dans la tradition néo-positiviste, le discours sur les causes. Il ne s'agit pas cependant de démontrer, comme dans le programme néo-positiviste, que les phénomènes peuvent être décrits par des lois universelles, mais de décrire *comment* telle régularité phénoménale résulte d'une structure interactive sous-jacente et des propriétés de ses parties. L'explication mécaniste se veut locale, attachée à répondre à une question « pourquoi » bien déterminée, concernant des phénomènes circonscrits. Aussi, selon une troisième interprétation, plus prudente encore, peut-elle être d'abord décrite comme une méthode d'analyse scientifique.

Notre thèse semble, à première vue, corroborée par les caractéristiques que prête Piccinini aux mécanismes. Celles-ci ressemblent en effet fortement à celles qu'on attribuerait à des programmes informatiques :

1. Les mécanismes sont caractérisés par des *états* (appelés aussi parfois propriétés) qui évoluent de conditions initiales à des conditions finales, comme

166. (Machamer, Darden et Craver 2000, p. 3). Une discussion des variantes à cette définition se trouve dans (Bechtel 2008, p. 13).

le sont aussi les machines de Turing et les programmes procéduraux en général ;

2. Les propriétés causales des mécanismes sont dues à l'organisation des interactions entre leurs parties et leurs activités, comme le sont les programmes complexes (comparer la définition des mécanismes ci-dessus, et la caractérisation de l'ordinateur par Simon (1996a, p. 17) : « Les seules propriétés détectables dans le comportement d'un ordinateur sont ses propriétés organisationnelles ») ;
3. Les parties et les activités de mécanismes s'analysent elles-mêmes en sous-mécanismes jusqu'à arriver à des constituants élémentaires (par exemple chimiques, pour les sciences de la vie), comme un programme s'analyse en sous-programmes jusqu'à arriver aux opérations élémentaires du langage utilisé ; les explications mécanistes sont, comme les programmes informatiques, organisées en hiérarchies descriptives emboîtées causalement [167].

Cependant, les mécanismes présentent des caractéristiques qui les rendent bien plus génériques que des programmes informatiques. Nous les décrivons négativement, par généralité croissante, en reprenant les distinctions établies par Piccinini :

1. D'abord, à part justement les programmes, peu de mécanismes se présentent comme l'exécution d'une procédure ou d'un code qui la représenterait au sein du système. On peut arguer qu'appartiennent à ce cas de figure les mécanismes génétiques (synthèse de protéines à partir de l'ADN, réplication de l'ADN) mais une telle configuration reste exceptionnelle. Non seulement les mécanismes naturels, mais également la plupart des artefacts mécaniques ne se présentent pas comme l'exécution d'un code, à commencer par les portes logiques elles-mêmes.
2. Les états et les activités des mécanismes peuvent être des grandeurs continues qui changent dans un temps continu, et de manière non nécessairement synchronisée, contrairement à celles d'un ordinateur qui évoluent par pas de temps discret, synchronisées par une horloge, et dont les états sont définis par des symboles ou des grandeurs numériques discrètes.
3. Contrairement aux procédures formelles, dont l'exécution consiste en un traitement symbolique indifférent au substrat matériel de ces symboles, le bon fonctionnement des mécanismes n'est pas en général indépendant des

167. Comparer (Machamer, Darden et Craver 2000, p. 13) et (Simon 1996a, p. 184). Sur la notion de constituant élémentaire voir (Machamer, Darden et Craver 2000, p. 15).

propriétés de leurs véhicules matériels ultimes, et leurs parties ne peuvent, par conséquent, être non plus assimilés à des symboles. Par exemple le mécanisme de la digestion, pour son bon fonctionnement, requiert que les aliments soient d'une certaine nature chimique.

4. Les mécanismes n'ont pas même nécessairement une fonction déterminée, ils ne se présentent pas comme une résolution de problème – ainsi les mouvements des astres.

La thèse générale de Piccinini est que le calcul (*computation*) est une espèce déterminée de mécanisme. Plus précisément, Piccinini définit le calcul à la frontière entre la seconde et la troisième différence. Ses deux propriétés essentielles sont donc d'être, selon lui (en reprenant la numérotation précédente) :

3. Indépendant pour son opération des substrats matériels qui constituent ses parties. Il opère sur des grandeurs (continues ou discrètes) qui sont réalisées comme des degrés de liberté de ces substrats, par exemple une intensité de courant électrique, une vitesse de rotation, etc. Piccinini les appelle également *indépendants de leur medium* (*medium-independent*). Une addition, par exemple, donne le même résultat que les nombres soient représentés par des bâtons, par des images mentales, par des engrenages de la machine de Babbage ou par les positions binaires de circuits électroniques.
4. Fonctionnel, c'est-à-dire qu'il existe une correspondance entre certaines grandeurs que l'on doit pouvoir localiser dans le système initial, et d'autres grandeurs également localisables dans la situation finale[168].

Par contre, selon Piccinini, cette notion de *computation physique* reste bien plus large que l'exécution d'un programme informatique. Il peut y avoir en effet :

1. Des calculs symboliques qui n'exécutent pas de programmes, – comme un circuit de portes logiques, dont le fonctionnement dépend des seules connexions des portes entre elles, et non d'un quelconque code, ou comme un algorithme statistique dont le comportement fonctionnel peut n'être pas déterminé par une procédure mais par des paramètres calibrés grâce à des données d'entraînement.
2. Des calculs non symboliques, comme les calculs effectués par les ordinateurs analogiques, ou par les systèmes de neurones (au sens propre) qui suivent, selon Piccinini, des lois propres, irréductibles à une description discrète aussi bien qu'analogique.

168. (Piccinini 2015, p. 121-123).

Ces distinctions de Piccinini sont extrêmement utiles pour notre propos. Elles vont permettre d'asseoir notre thèse principale, que la réflexion procédurale, loin d'être une espèce mineure et étroite de l'explication mécaniste, en est plutôt coextensive, et en fait, constitutive. En d'autres termes, nous souhaitons montrer que si nous décrivons le réel par des mécanismes, c'est *parce que* nous sommes engagés dans une investigation systématique à son propos, qui exige de l'exprimer par des significations procédurales (mathématiques). Cela permet de généraliser le résultat de la section précédente concernant les mécanismes de transmission : Si des procédures peuvent être inscrites dans des mécanismes, c'est parce que ceux-ci sont d'emblée des concepts procéduraux – parce que dans la modification technique d'un mécanisme, on ne fait que manipuler ce qu'on avait déjà identifié dans la nature de conforme à notre manière procédurale de penser.

§ 212. Procédure et esquisse de mécanisme

L'examen des deux premières distinctions est assez direct, puisqu'il s'agit simplement de constater que notre concept de *procédure* est bien plus large que celui de Piccinini.

Certainement, les portes logiques et les circuits imprimés n'exécutent pas de programme, si Piccinini a à l'esprit l'architecture de l'ordinateur de von Neumann, où une unité de contrôle centrale va chercher les instructions à exécuter dans une mémoire séparée. Mais l'ENIAC était programmé par câblage direct de ses composants entre eux, et l'implémentation des programmes élémentaires des processeurs sont encore aujourd'hui directement « imprimés » sur des circuits de transistors. Ces mécanismes sont l'œuvre d'une démarche procédurale, même si la procédure n'y est pas lisible directement.

La seconde distinction concerne les mécanismes de calcul analogiques, ou plus généralement qui impliquent des interactions non linéaires entre leurs parties. La nature essentiellement discrète des descriptions procédurales semble les rendre incapables de décrire ces mécanismes, dont les systèmes d'équations différentielles sont l'expression canonique.

Or il faut distinguer ici deux choses. Si on croit que l'objet d'une description procédurale est toujours un mécanisme de transmission (parce qu'on a à l'esprit les programmes informatiques et l'architecture de l'ordinateur moderne), alors bien entendu, de nombreux mécanismes ne peuvent être décrits de cette manière. Cependant, notre concept de *moyen procédural* est bien plus large que cela.

Une équation différentielle, comme l'équation de la balle rebondissante dans Modelica[169] peut bien manipuler des grandeurs continues, elle n'est cependant rien d'autre qu'une représentation procédurale indiquant, comme nous l'avons dit[170], quelles opérations (mesures, calculs, expérimentations, etc.) il est possible de réaliser pour expliquer, prédire ou contrôler le rebond de la balle. Il en est de même pour tous les systèmes complexes dont nous connaissons les lois locales d'interaction sans être capable cependant de les intégrer en une loi générale. De tels mécanismes chaotiques n'en sont pas moins décrits de manière procédurale, puisque leur contenu positif dit ce qu'il est tout de même possible d'inférer localement du phénomène observé.

Autrement dit encore, en tant qu'un mécanisme est constitué d'interactions entre ses parties, ce sont ces relations qui sont représentées, et celles-ci ne sont rien d'autres que des règles procédurales qui nous disent *ce qu'on peut attendre du comportement d'un tel mécanisme*, et qu'on doit pouvoir définir exactement (mathématiquement). Dans le cas contraire – si on avait affaire à des relations floues ou approximatives, on ne parlerait tout simplement pas de mécanisme. On retrouve ici l'argumentation de Dowek : un objet d'interactions réglées est d'abord un objet mathématique.

La troisième distinction concerne les mécanismes que, selon la définition de Piccinini, on ne doit plus appeler computationnels. Un mécanisme en effet, ne se réduit pas toujours à de pures interactions entre ses parties dans l'indifférence aux substrats matériels de celles-ci. Ils peuvent au contraire en dépendre de manière cruciale. L'exemple paradigmatique, déjà cité, est la digestion[171]. C'est parce que les glandes sécrètent telles substances chimiques que l'estomac est capable de digérer tels aliments. Une procédure effective étant, au sens où nous l'entendons ici, symbolique, ou encore mathématique, elle ne peut donc représenter ces mécanismes, elle peut au mieux les simuler, en remplaçant les matériaux, et les actions en question, par des symboles.

Cependant, pour qu'une explication mécaniste soit complète, il faudrait encore expliquer pourquoi tel matériau est important pour tel mécanisme, quelles sont les raisons pour lesquelles il est nécessaire à son fonctionnement. On peut dire que les substances chimiques ont le « pouvoir » de dissoudre certaines liai-

169. Voir plus haut, § 37.
170. Voir plus haut, § 176.
171. (Piccinini 2015, p. 146).

sons moléculaires des aliments, mais ce « pouvoir », ou cette « propriété » doit être elle aussi expliquée. Si ce n'est pas le cas (si les chimistes ne savent pas fournir un tel résultat), on a affaire à une *esquisse de mécanisme*, c'est-à-dire à une explication partielle qui admet certaines propriétés élémentaires à ses composants sans les démontrer. Pour parvenir à une explication complète, ces propriétés et activités devraient elles-mêmes être expliquées en mécanismes plus élémentaires, ici au niveau chimique.

Une telle différence entre mécanisme et esquisse de mécanisme ne concerne pas le raisonnement lui-même, mais seulement ses prémisses ; dans le langage de Dewey, il ne porte pas sur le moyen procédural utilisé, mais sur ses moyens matériels, qui ne sont pas attestés de manière certaine. La confirmation *a posteriori* de l'hypothèse ne changera rien à l'explication mécaniste elle-même, mais seulement au degré de confiance que nous lui accorderons.

Ainsi, ce que les nouveaux mécanistes appellent *esquisse de mécanisme* désigne les moyens procéduraux de l'explication, c'est-à-dire ce qui peut proprement être mathématisé. Une esquisse de mécanisme est d'emblée explicative parce que dans le système dans lequel elle est explicitée (dans notre exemple, celui du gastro-entérologue, et non celui du chimiste) elle est un moyen procédural (un ensemble de relations) construit à partir de moyens matériels admis (les règles d'inférence admises concernant la chimie des aliments) pour permettre à l'investigation d'expliquer les phénomènes, de les ramener à une plus grande unité de compréhension, et rendre possible leur contrôle.

Poser que toute esquisse de mécanisme peut en droit être développée jusqu'à s'exprimer dans les termes de certains mécanismes élémentaires fondamentaux relève d'une posture réaliste, qui recherche l'adéquation entre les choses et les représentations. On *postule* que les choses sont *réellement organisées* ainsi que le décrit le mécanisme, et *réellement étagées* en plusieurs niveaux d'organisation. Piccinini opère explicitement une telle supposition[172]. Craver, un autre philosophe mécaniste proche de Piccinini est plus nuancé :

> Notre orientation […] pourrait être décrite comme un réalisme de bon sens modéré par un pragmatisme raisonnable. L'idée même que des esquisses de mécanisme sont évaluées en terme de leur *correction* suppose qu'il existe un mécanisme cible, entièrement instancié dans le monde, et que l'on peut évaluer le degré d'adéquation ou de correspondance entre l'esquisse et la cible. L'idée même que des esquisses

172. (Piccinini 2015, p. 105).

> de mécanisme sont évaluées en terme de leur *complétude* suppose qu'il existe un mécanisme cible complet dans le monde, et que l'on peut évaluer la mesure dans laquelle l'esquisse inclut exactement toutes les entités, activités, et propriétés organisationnelles dans la cible. En nous exprimant ainsi, nous entendons admettre qu'une esquisse peut être suffisamment complète et correcte *pour l'objectif en jeu*, sans l'être entièrement. On peut accepter les idéaux de complétude et de correction pour les descriptions de mécanisme et, dans le même temps, reconnaître que la science a affaire avec des esquisses incomplètes et idéalisées[173]. *(Nous soulignons)*

Dans ce passage Craver semble vouloir conjurer l'hypothèse fictionnaliste sur laquelle Descartes avait fondé sa physique mécanique. Nos esquisses gagneraient une certitude croissante au fur et à mesure que nous en expliciterions les mécanismes sous-jacents, qui « colleraient » de mieux en mieux au réel à chaque fois qu'un niveau d'analyse inférieur serait conquis. Cette idée évoque les métaphores du peintre qui affine progressivement son esquisse en y ajoutant des détails jusqu'à ce que son tableau soit « plus vrai que nature », ou du tailleur qui ajuste l'habit progressivement à son modèle, au cours de séances successives d'habillage. Mais de telles images ont pour contre-effet de déclasser toute explication mécaniste concrète au niveau d'une simple validité pragmatique, dépendant de « l'objectif en jeu ».

Par ailleurs, Craver se garde bien de formuler son « mécanisme cible » sous la forme d'une assertion réaliste. Il se contente d'en faire un idéal régulateur de l'investigation. Il ne peut donc ni servir, ni contraindre l'élucidation. Une explication mécaniste n'est elle-même que la récapitulation, dans un certain système symbolique, d'une investigation scientifique. Elle permet, sans avoir à réitérer l'investigation, d'interagir de manière efficace avec le phénomène considéré, soit pour expliquer telle ou telle observation, soit pour la prédire, soit pour la contrôler. Elle est donc elle-même une procédure ou un ensemble de procédures, dont l'exécution se résout dans les sous-procédures que commandent les symboles manipulés.

C'est donc remettre les choses à l'endroit qu'affirmer qu'une explication mécaniste, si un réaliste peut certes la voir comme un pointeur plus ou moins distant des choses mêmes, est d'abord et avant tout une machine inférentielle, qui possède autant de certitude qu'en ont ses composants élémentaires. Les termes mêmes employés par Craver dans le passage cité, *complétude* et *correction*, indiquent bien que l'enjeu central d'un mécanisme est de développer des chaînes d'inférence de

173. (Craver et Darden 2013, p. 9).

manière systématique. La précision de la décomposition n'est pas nécessaire à ces fins, mais elle permet de les confirmer. « Le mécanisme cible », selon l'expression de Craver, loin de désigner la chose réelle, semble plutôt jouer le rôle d'une norme idéale ou régulatrice, au sens de Kant, qui exige que l'explication soit procédurale de part en part, c'est-à-dire que ses moyens matériels puissent chaque fois être explicités de manière procédurale eux-mêmes, jusqu'à parvenir à des particules élémentaires dont les lois causales ne pourraient plus être remises en cause.

La position de Craver est conforme à l'épistémologie de la science qui se met en place après Newton[174], dont nous avons déjà relevé, suivant Peter Dear, l'ambiguïté fondamentale. Expliquer les phénomènes en dépliant successivement tous leurs mécanismes sous-jacents, c'est se situer dans l'orbe de la question proprement théorique du *« Qu'est-ce que c'est ? »*, ce qui suppose une attitude réaliste, la confiance dans l'adéquation possible, d'une manière ou d'une autre, des choses et des représentations. Mais ces mécanismes sont tout aussi bien des procédures de vérification d'inférences portant sur les phénomènes, qui sont mises à la disposition de toute sorte de résolution de problèmes où ils sont impliqués, qu'il s'agisse de les prédire ou de les contrôler.

§ 213. L'argument téléologique

La dernière différence entre procédure et mécanisme porte sur la dimension fonctionnelle des procédures : une procédure est définie pour résoudre un problème, tandis que la description d'un mécanisme ne suppose pas en général de lui identifier une finalité quelconque. Comme le dit Piccinini, le déplacement d'une galaxie n'est pas un mécanisme téléologique[175]. Cependant nous avons vu que d'une manière générale, la notion de fonction elle-même était pour les mécanistes une notion épistémique provisoire, amenée à se résoudre en sous-mécanismes. Cette différence ne semble donc pas pouvoir être absolue.

Piccinini tente cependant de lui donner un fondement objectif à partir de la notion d'organisme vivant, qui arbore deux mécanismes spécifiques : celui de conservation et celui de reproduction. Il propose de définir comme fonctionnels tous les mécanismes qui leurs sont nécessaires ou qui les favorisent. Les mécanismes artificiels, conçus par les humains, sont donc fonctionnels par ce biais. On voit d'emblée les difficultés auxquelles cette approche naturaliste est confrontée. Piccinini doit

174. Voir plus haut, § 125.
175. (Piccinini 2015, p. 100).

par exemple rendre compte des artefacts néfastes aux personnes et à l'espèce humaine – la drogue, le jeu, etc. Mais il est encore plus difficile de justifier que la notion de fonctionnalité varie selon les circonstances. Une voiture serait fonctionnelle, puisqu'elle permet d'augmenter la puissance de déplacement des personnes, mais les voitures de compétition automobile ne le seraient pas. Piccinini est ainsi forcé d'introduire la notion de fonction *subjective*, elle-même inexpliquée[176], et qui annihile ses efforts fondationnels précédents.

Il est intéressant de remarquer que Piccinini, dans deux chapitres terminaux consacrés à ce qu'il appelle la thèse de Church-Turing physique, définit le critère fonctionnel par un autre biais. En argumentant contre la forme « ambitieuse » (*bold*) de cette thèse, qui affirme que « tout processus physique est calculable au sens de Turing[177] » Piccinini introduit une contrainte d'utilisabilité (*usability*) par un observateur fini du processus. Il faut que celui-ci puisse *s'en servir* pour calculer, et donc que celui-ci soit exécutable à souhait, fiable, automatique, et uniforme (il ne doit pas devoir être modifié pour chaque entrée différente). L'exécutabilité, à son tour, est analysée par un critère d'indépendance à l'égard du processus physique (on doit pouvoir calculer autre chose que la situation finale du processus lui-même), par la capacité de l'observateur à déterminer les entrées du processus et à en lire les résultats d'intérêt, et enfin par un critère de constructibilité. Le premier critère correspond à celui, examiné précédemment, d'indépendance du calcul relativement aux substrats matériels du processus physique[178]. Le troisième critère est étrange ; il semble limiter la capacité de calcul à des systèmes artificiels. Mais telle n'est pas l'intention de Piccinini. L'utilisabilité d'un système implique sa constructibilité au sens où, *a minima*, un dispositif doit pouvoir être mis en place qui permette de renseigner les données du calcul et de lire ses résultats. Ainsi prend-il l'exemple suivant, un peu fantaisiste, d'un dispositif calculatoire impliquant le soleil lui-même, ainsi qu'un canon spatial :

> Supposons que tirer n impulsions d'énergie vers une tache solaire de manière fiable et répétée provoque l'émission de $f(n)$ impulsions de rayonnement, pour une certaine fonction f non triviale. Le système qui comprend les taches solaires ainsi que l'appareil pour émettre et recevoir des impulsions d'énergie est physiquement constructible au sens entendu ici. Il en est plus généralement ainsi de toute techno-

176. (ibid., p. 113).
177. (ibid., p. 251).
178. (ibid., p. 256).

logie de calcul qui exploite les propriétés naturelles des matériaux existants[179].

La constructibilité ne concerne donc pas nécessairement la machine elle-même qui calcule, mais la « technologie de calcul », c'est-à-dire le dispositif qui va rendre possible la *procédure* par laquelle l'observateur va interagir, en entrées et en sortie, avec elle. Cette procédure constitue le second critère de l'exécutabilité, que Piccinini identifie au critère fonctionnel[180]. Un mécanisme est donc fonctionnel si un observateur peut s'en servir pour calculer, en particulier déterminer ses entrées et lire ses résultats. Piccinini reste flou sur la nature de cet observateur : il mentionne les êtres humains ou tout être intelligent ayant des capacités équivalentes, mais évoque la possibilité de comprendre cet observateur de manière plus large encore, en incluant « tout système organisé fonctionnellement dont le comportement est influencé par un calcul d'une manière appropriée ». Aussi, dit-il, si les systèmes nerveux calculent (ce que Piccinini croit) on pourrait considérer comme observateurs « les corps des organismes influencés par leurs propres calculs neuronaux[181] » ! L'étrangeté de ces expressions montre tout simplement qu'il n'est pas possible d'abstraire complètement le sujet (humain) de l'investigation qui établit la description procédurale. Même lorsque celle-ci semble la plus objective possible, dans sa forme mécaniste, elle se réfère ultimement à des actions possibles par ce sujet, ou par d'autres personnes ou machines qu'il aura préposées à cet effet.

Aussi parler de mécanismes qui n'ont pas de fonction ou de finalité ne peut se comprendre que relativement à la visée théorique de l'investigation. On se rappelle en effet que nous avions distingué deux attitudes de la réflexion, qui éclairent selon nous l'opposition traditionnelle entre pratique et théorie, la première occupée à résoudre des problèmes particuliers, et l'autre visant à contrôler par avance la validité des inférences qu'elle utilise dans ces recherches, et réorganisant en conséquence les systèmes de significations qui leur sont sous-jacents[182]. Il y a interdépendance et coopération de l'une avec l'autre. la résolution de problème bénéficie grandement du « travail de terrassement » qu'effectue la connaissance théorique, car elle lui évite de mener quantités de vérifications par elle-même, et elle lui procure des concepts adéquats à toute situation. Réciproquement, toute recherche théorique de contrôle d'une inférence ou de réorganisation conceptuelle n'est rien d'autre qu'un problème particulier à résoudre – l'investigation théorique *se réalise*

179. (Piccinini 2015, p. 253).
180. (ibid., p. 256).
181. (ibid., p. 250).
182. Voir plus haut, section 10.4.

à travers la résolution de problème.

Les explications mécanistes sont de ce type : il s'agit de résolution de problèmes *théoriques*, c'est-à-dire de problèmes posés par un investigateur visant à vérifier et à organiser de manière adéquate un réseau d'inférences à propos d'un domaine du réel. Cela implique qu'au sein de l'investigation, l'insistance n'est pas tant portée sur la recherche d'une solution d'un problème précis, qu'à la constitution d'un espace de problème adéquat pour exprimer une large classe de problèmes portant sur un objet donné[183].

Un mécanisme apparaît ne pas avoir de finalité car il est présenté sous une forme théorique. Mais sous cette forme, toute chose s'en dépouille, aussi bien le mouvement des galaxies que la propagation des virus ou les actions humaines. En tant qu'un mécanisme n'est rien d'autre qu'un bloc procédural d'inférences, son indifférence aux fins signifie simplement son universelle disponibilité pour tout type d'investigations, qu'elles soient à visée explicative, prédictive ou technique[184]. Il n'y a pas d'objets qui seraient intrinsèquement fonctionnels, comme les meubles, et d'autres qui ne le seraient pas, comme des bouts de bois, mais seulement des réseaux de règles associées aux objets. Ces règles, en tant qu'on les mobilise dans une résolution locale de problème, apparaissent comme utilitaires ; en tant qu'elles sont seulement présentées dans le cadre d'un corpus théorique, elles apparaissent n'avoir aucune fonction.

§ 214. Le privilège du mécanisme de transmission

Contrairement à ce qu'avance Piccinini, ce qui distingue les procédures des explications mécanistes n'est donc pas qu'elles exécutent des instructions, ni qu'elles procèdent de manière séquentielle et discrète, ni encore qu'elles soient indépendantes d'un substrat et non plus enfin qu'elles sont fonctionnelles et les autres non. Elles sont, aussi bien que les mécanismes, des réseaux d'inférences qui permettent de résoudre des problèmes locaux. Mais les problèmes que résolvent les explications mécanistes sont théoriques, c'est-à-dire ordonnés à la simple visée de leur organisation systématique en vue de leur vérification et de leur mise à disposition pour toute résolution de problème, ce qui leur donne leur caractère neutre et non-final.

183. Voir plus haut, § 181.
184. Voir plus haut, § 181.

Par la critique de ces quatre différences que Piccinini établit entre procédure et mécanisme, nous renforçons donc la thèse de Dowek que ces notions sont co-extensives. Il nous faut donc à présent revenir au pan-computationalisme qui affirme le privilège des mécanismes de transmission – informationnels – parmi toutes les classes de mécanismes.

S'il est possible qu'une procédure soit inscrite dans un mécanisme et exécutée par lui, c'est parce qu'un mécanisme n'est rien d'autre qu'une procédure interprétée comme signe immédiat, c'est-à-dire comme moyen matériel, d'un objet dynamique de l'expérience. On se meut toujours ici dans le plan des significations. L'« inscription » d'une procédure dans un mécanisme est une séquence d'actes de l'investigateur sur cet objet, en tant qu'il est décrit dans un système de significations donné (l'explication mécaniste) qui lui-même prévoit telles et telles conséquences de ces actes. Que ces actes d'inscription aient pour conséquence l'exécution réussie, par le mécanisme ainsi modifié, de la procédure visée n'est pas le signe d'une quelconque *harmonie préétablie* du corps et de l'esprit[185] mais simplement une confirmation complémentaire et provisionnelle de la pertinence du système de significations dans lequel nous interprétons notre expérience et décidons d'agir.

Ce raisonnement peut sembler aboutir à un paradoxe, ou à l'une de ces *boucles étranges* (*strange loops*) du raisonnement étudiées par Hofstadter : si tout mécanisme n'est que le corrélat objectif d'une procédure explicative, et que toute procédure est en droit réalisable dans un mécanisme de transmission, cela ne signifie-il pas que le réel n'est explicable que dans la mesure où on peut le représenter par des transmissions de signaux, c'est-à-dire dans la mesure exacte où il peut être décrit dans les termes de la théorie de l'information – ce qui reviendrait à l'assertion d'un pan-computationnalisme au moins épistémique ?

Il ne s'agit pas ici de discuter la portée de la théorie physique de l'information, ce qui est l'affaire des scientifiques. Mais quand bien même cette théorie serait la forme la plus élégante et la plus directe d'expression des lois fondamentales de la nature, et quand bien même elle résoudrait tous les problèmes connus de physique fondamentale, cela n'aurait rien à voir avec ce dont nous parlons ici.

Si en effet toute forme de mécanisme peut être simulée par un mécanisme de transmission, c'est parce que tout espace systématique de significations (fini ou plus généralement inductif) peut être codé dans le système infini élémentaire des entiers naturels, et qu'il existe des mécanismes de transmission permettant de coder

185. Voir plus haut, § 3.

toutes les fonctions calculables sur $\mathbb{N}$[186].

Or cette dernière propriété (appelée complétude au sens de Turing) *n'est pas spécifique* aux mécanismes de transmission.

Les ordinateurs analogiques sont un autre cas simple de machines programmables qui ne fonctionnent pas par contrôle de transmission. À première vue, ils ressemblent beaucoup aux ordinateurs tels que nous les connaissons, puisqu'ils se présentent comme des circuits de composants électroniques alimentés par un flux électrique homogène. Leur programmation procède comme celle de circuits booléens, par construction de circuits. Cependant, ces composants n'ont pas une simple fonction de transmission du signal, comme les portes logiques. Ils sont actifs, en ce qu'ils vont *transformer* le signal – et ainsi par exemple une capacité va permettre de représenter l'*intégration* mathématique d'une valeur. Cette opération ne peut être représentée par un mécanisme de transmission – puisqu'il porte sur des valeurs continues. Il est possible de construire des ordinateurs analogiques universels[187].

Il en est de même pour les ordinateurs quantiques, qui permettent de calculer des fonctions sur des qubits* à l'aide de portes quantiques* et de l'intrication quantique. On a également montré la possibilité de programmer des machines bio-chimiques[188] et bactériennes[189]. Des matériaux déformables (par exemple à mémoire de forme) pourraient être utilisés pour réaliser des ordinateurs neuromorphiques[190], dont on a démontré la complétude au sens de Turing[191].

Toutes ces dernières recherches sont encore largement à l'état conceptuel. Cela est cependant suffisant pour notre propos, qui est de souligner que la notion de programmation est plus large que celle d'ordinateur *à contrôle de transmission* tel que nous le connaissons aujourd'hui, et que plusieurs nouveaux paradigmes d'ordinateurs pourraient émerger à l'avenir, dont nous n'avons pas encore idée. Un ordinateur n'est rien d'autre qu'un phénomène dynamique de l'expérience qu'on sait modifier de manière à y référer les opérations élémentaires (les moyens matériels) d'un système de significations mathématiques, ainsi que leurs compositions possibles (les moyens procéduraux). Ces moyens matériels peuvent signifier autre

186. Voir plus haut, § 206.
187. (Graça et Costa 2003).
188. (Fages et al. 2017).
189. (Becerra, Gutiérrez et Lahoz-Beltra 2022).
190. (Kaspar et al. 2021, p. 352).
191. (Date et al. 2022).

chose que des transmissions. C'est en ce sens qu'il faut interpréter la distinction 2) que fait Piccinini, entre calculs numériques (ou discrets) et calculs en général, si on entend par là qu'il est possible de reconnaître, dans d'autres formes de mécanisme, des moyens procéduraux et de les manipuler.

Il n'y a donc nulle raison d'accorder un privilège ontologique à l'information, qui n'est que l'abstraction procédurale du flux d'une transmission. On peut bien sûr développer, pour d'autres raisons, une métaphysique computationnelle, mais l'effectivité de nos ordinateurs actuels ne pourrait nullement être mobilisée comme argument en sa faveur.

Si d'autres modèles de programmation sont en droit possibles, d'où vient la prédominance actuelle du contrôle de transmissions ? Une réponse aisée est qu'il s'agit du mécanisme le plus *antique*, celui qui fut historiquement développé jusqu'à rendre pensable le concept de programmation. La programmation d'ordinateurs est, de ce point de vue, l'aboutissement d'une compétence bien plus ancienne – celle de la construction de mécanismes de transmission – qui commence sans doute avec les premiers systèmes d'irrigation, se continue avec les clepsydres et autres automates hydrauliques, et prend son essor enfin avec l'apparition des montres mécaniques au XIII^{e} siècle puis des mécanismes industriels.

Cette explication historique n'est cependant pas entièrement correcte, car les ordinateurs analogiques furent conçus antérieurement aux ordinateurs numériques. S'ils ne parvinrent pas se développer comme les seconds, au point d'être largement réduits aujourd'hui au statut de curiosités technologiques et conceptuelles, c'est que, s'ils sont très adaptés à la représentation d'équations différentielles, ils sont cependant moins aisés à manipuler pour représenter tout type de procédures.

Or c'est cela qui est le point de départ de la programmation. Le programmeur ne sait certes en général pas à quoi ressemblera exactement le mécanisme de transmission correspondant à son programme, mais il a la présomption que celui-ci, par un jeu plus ou moins complexe de traductions intermédiaires, entrera *in fine* dans une correspondance directe et locale (« point à point ») avec des inférences réalisables sur l'état de sa machine.

Les ordinateurs biologiques, analogiques et quantiques n'exhibent pas une telle propriété, tout en étant programmables. Un programme quantique peut être présenté comme une séquence de *portes* modifiant l'état et l'intrication de qubit*, et cette séquence est donc procédurale, et se code d'ailleurs de manière classique,

grâce à des librairies étendant des langages de programmation réguliers comme Python. L'exécution de cette procédure, cependant, ne peut entrer en correspondance avec les termes de celle-ci de la même manière qu'avec un mécanisme de transmission. Les inférences, même élémentaires, qu'on peut tirer de ce dispositif doivent faire l'objet de calculs complémentaires, à tout le moins parce que ces inférences sont probabilistes et intriquées.

Autrement dit encore, le privilège des mécanismes de transmission parmi tous les autres phénomènes dynamiques est qu'ils peuvent représenter de manière *transparente* l'inférence d'une expérience A à une expérience B, du fait même que l'écoulement d'un fluide d'un point A à un point B, ou la transmission de mouvement d'un objet solide A à un autre B, c'est-à-dire l'action par contact, *sont parmi les inférences les plus primitives et simples que nous puissions faire* dans l'expérience. C'est notamment grâce à l'action par contact que Hume, avec l'exemple des boules de billard, explicite son principe d'association[192]. Elle est une sorte d'idéal régulateur de l'explication, qui permet d'exprimer directement l'inférence faite dans l'expérience par une relation symbolique.

Il y a ainsi dans le terme de *mécanisme* l'idée d'une transparence de l'explication à l'expérience vécue, d'une adéquation immédiate de la relation signifiante et de l'inférence, qui ferait qu'un symbole mathématique dénote immédiatement un changement, et qui serait donc éminemment constructible, à l'instar d'une pure figure géométrique. L'idée de mécanisme traduit l'intention de décrire le monde de manière mathématique, c'est-à-dire comme résultat d'un enchaînement procédural d'opérations pures – comme on décrit les artefacts qu'on a soi-même construits.

C'est dans cette intuition, peut-on avancer, que serait ancrée l'évidence des doctrines mécanistes de la nature, qu'elles soient atomistes ou non : que toute explication des phénomènes dynamiques doit pouvoir être ultimement réduite à des transmissions de mouvement, et donc être exprimée dans les termes de la seule géométrie. E. J. Dijksterhuis (1986) conclut son histoire de la mécanique en affirmant que le développement de cette science n'est rien d'autre que celui de l'esprit scientifique, qui entreprend la mathématisation du savoir *sur le fond du présupposé que le monde est descriptible mécaniquement*. Autrement dit, la certitude que les phénomènes sont mathématisables proviendrait de ce qu'ils seraient descriptibles comme des agencements de mécanismes. Cette liaison est rendue évidente par nos développements, car nous avons suffisamment montré que les mécanismes

192. (Hume [1740] 1990, p. 649).

ne sont rien d'autre que des dispositifs procéduraux purs, *c'est-à-dire* exprimables mathématiquement. La *philosophie mécanique* n'est rien d'autre que l'expression d'une exigence de transparence absolue de l'explication, exigence dont le pan-computationnalisme est bien un avatar contemporain.

11.6 Ratiocination et investigation

Nous avons, au cours de la section précédente, mis de côté deux interprétations de la notion de mécanisme qui la considéraient comme un phénomène de l'expérience. La première (pan-computationaliste), reconnaissant leur caractère procédural, identifiait tous les mécanismes au modèle dominant du contrôle de transmission. La seconde (néo-mécaniste), reconnaissant que ce dernier n'était qu'un type de mécanisme parmi d'autres, tentait de distinguer les notions de *procédure* et de *mécanisme*.

Si ces interprétations sont défaillantes, c'est qu'elles manquent de voir que machines et mécanismes sont d'abord des catégories de l'investigation, qui désignent des objets de l'expérience *à propos desquels on juge qu'une ratiocination systématique est possible*. L'idée d'une ratiocination systématique – c'est-à-dire d'une investigation se résumant strictement à l'application de règles prescrites dans un système – est centrale dans le projet de connaissance qui se développe au cours des Temps modernes à partir du noyau constitué par la géométrie et le mécanisme. Nous avons suffisamment insisté sur le fait qu'il constituait une transformation importante du concept de connaissance humaine. À bien des égards, cette transformation fait même *violence* aux pratiques épistémiques courantes des personnes, ce qui se voit par exemple dans l'exigence faite à l'investigateur de se comporter *mécaniquement* lors de ces ratiocinations[193].

Il est temps à présent de reprendre l'examen des apports de l'informatique à ce projet de connaissance. Comme nous l'avions déjà noté[194], la réponse peut sembler évidente : en automatisant les ratiocinations, la programmation permet de réaliser des gains de productivité sur l'investigation, puisque les ratiocinations ne sont rien d'autres que des investigations, si on peut dire, « mises en boîte ». Il n'y a qu'à évoquer ici tous les calculs que réalisent les ordinateurs pour les scientifiques, et qui furent l'une des premières motivations de l'invention et du progrès de l'in-

193. Voir plus haut, § 202.
194. Voir plus haut, § 209.

formatique. Nous avons cependant immédiatement pointé qu'une telle explication pourrait rester superficielle, en faisant nôtre la remarque de Paul Humphreys[195], selon laquelle les progrès de la science moderne avaient régulièrement été corrélés à l'accroissement des capacités de calcul. D'une certaine manière, l'invention des ordinateurs n'a pas réduit la part du calcul dans la recherche scientifique – ce qu'on pourrait attendre en droit de simples gains de productivité – mais l'a au contraire augmentée, comme si elle avait ouvert de nouvelles possibilités à la connaissance, plutôt que réduit la part de ses tâches subalternes.

Notre idée principale est que, loin de se limiter à des gains de productivité, l'idée d'une *science automatisée*, avancée par Humphreys, fonctionne comme une sorte d'idéal régulateur du projet moderne d'une connaissance systématique. Comme tout idéal régulateur, il s'agit d'une norme *locale* de progrès et d'achèvement – celle de pouvoir récapituler toute investigation scientifiquement réussie par une procédure si exacte et exhaustive, que celle-ci pourrait être (en droit) vérifiée mécaniquement. Cette norme se déploie *de facto* dans les trois principaux domaines de la connaissance systématique – les mathématiques, les sciences et l'ingénierie, comme nous allons tenter de l'illustrer à l'aide de quelques exemples. Cependant, il faut bien noter que, comme tout idéal régulateur, une telle norme est inactive *globalement*, c'est-à-dire qu'elle est incapable de donner des directions concrètes à la recherche, parce que la ratiocination n'a jamais porté sur les deux moments essentiels de l'investigation, qui continuent de l'encadrer, et qui restent essentiellement humains : la reconnaissance initiale de la situation problématique et le jugement final qui la conclut.

Il ne peut s'agir dans le cadre de ce travail d'étayer cette thèse avec toute l'ampleur et la précision qu'elle requerrait. Nous commençons par présenter, sans prétention à l'exhaustivité, quelques exemples d'usages concrets de la programmation au sein des mathématiques (§ 215), des sciences (§ 216) et de l'ingénierie (au sens large), où nous développons de manière un peu plus suivie le cas particulier de l'architecture et de l'ingénierie civile (§ 217). Ces illustrations nous permettent, en conclusion, de mieux expliciter la thèse que nous venons d'avancer (§ 218).

195. Voir plus haut, p. 818.

§ 215. Les assistants de preuve en mathématiques

L'exemple le plus célèbre d'usage de l'ordinateur en mathématiques est la preuve du théorème dit *des quatre couleurs* par Appel et Haken en 1976[196]. La raison de cette célébrité est justifiée : non seulement cet exemple fut l'un des premiers requérant l'ordinateur pour exécuter la preuve, mais il permit de conclure une conjecture parmi les plus célèbres des mathématiques du siècle dernier, qui portait sur la possibilité de toujours colorier avec seulement quatre couleurs les différentes zones (ou « pays ») d'une carte quelconque, sans jamais que deux zones adjacentes aient la même couleur. Le contraste entre la simplicité de l'énoncé et la difficulté de la preuve en firent un sujet de recherche important. La stratégie d'Appel et Haken fut de raisonner par l'absurde sur une carte qui requerrait au moins cinq couleurs pour être coloriée. Elle consista à identifier d'abord un ensemble de plus de 1400 configurations dites « inévitables », c'est-à-dire dont l'une au moins devrait convenir à une telle carte, puis à montrer algorithmiquement que toutes ces configurations pouvaient être coloriées en quatre couleurs. L'ordinateur explora ces 1400 configurations une par une, en utilisant plus de 400 tactiques de réduction. Il est intéressant de noter ces commentaires d'Appel à propos de l'ordinateur, qui selon lui devint un véritable « partenaire » de l'équipe travaillant sur le théorème :

> Mon impression la plus pertinente sur ce sujet est que l'ordinateur ne pensait pas comme un mathématicien […] Il utilisait […] ces bouts de connaissance dont il disposait de toutes les manières possibles et imaginables, et n'importe quel mathématicien lui aurait dit : « Non, non, non, vous devez vous organiser, vous devez le faire de cette manière », mais l'ordinateur ne faisait pas ainsi. Et il eut beaucoup plus de succès, parce qu'il ne pensait pas comme un mathématicien[197].

On a ici un exemple éclatant d'un gain de productivité sur la ratiocination qui permet d'accéder à de nouveaux résultats, pratiquement inaccessibles à la ratiocination humaine, ce qui rejoint la remarque de Babbage que certaines opérations de machines industrielles présentent une force, une vitesse, ou une précision telles qu'on ne peut plus vraiment mesurer leur apport en gain de productivité, tout simplement parce qu'elles sont incommensurables avec ce que pourraient réaliser des travailleurs humains[198]. Dans ces cas de figure, l'investigation ne consiste plus à

196. Nous nous basons ici sur la présentation qu'en fait (MacKenzie 2004, ch. 4).
197. Entretien avec Appel, (ibid., p. 133).
198. Voir plus haut, § 88.

opérer ces ratiocinations une à une, mais à les préparer et à les agencer pour leur exécution mécanique ; elle devient architecte, et non plus ouvrière de la ratiocination.

Comme le résume De Mol, cette méthode mit en débat les normes jusqu'ici largement tacites de la démonstration mathématique, à savoir leur capacité à être inspectées par des personnes humaines, à être compréhensibles pour elles, et enfin à ne pas dépendre de dispositifs empiriques faillibles comme un ordinateur ou un logiciel peuvent l'être[199].

Selon De Mol, ces questions doivent être abordées à partir de ce que l'ordinateur change à la *pratique investigatrice* du mathématicien. Elle étudie pour ce faire un autre exemple de preuve assistée par ordinateur, concernant un problème indécidable classique de la théorie de la calculabilité. Il s'agit du problème des *castors affairés* (*busy beavers*), image anglo-saxonne d'ouvriers industrieux, prise ici comme métaphore des machines de Turing binaires imprimant un nombre maximum de 1 sur un ruban initialement vierge avant de s'arrêter[200]. Le problème est de trouver, pour une machine ayant n états, ce nombre maximum. Il est indécidable en général, mais peut être solvable pour des n particuliers ; il l'a été pour n égal à 2, 3 ou 4. Dans ces deux derniers cas, l'espace de problème (nombre de machines de Turing binaires à 3 ou 4 états) est respectivement supérieur à 4,8 millions de possibilités et à 6,9 milliards. La solution de ces problèmes est qu'une machine à 3 états peut au mieux imprimer six 1 avant de s'arrêter, et une machine à 4 états treize.

De Mol décrit les techniques utilisées par les mathématiciens pour trouver ces résultats. Le raisonnement permet d'éliminer une grande majorité de cas, mais il en reste néanmoins 4000 pour $n = 3$ et plus de 600000 pour $n = 4$. La recherche devient alors heuristique : il faut définir des stratégies qui permettent d'éliminer des possibilités le plus vite possible afin de limiter la simulation effective de machines de Turing aux candidats les plus intéressants. Les mathématiciens mobilisèrent naturellement des ordinateurs, non seulement pour cette dernière tâche, mais également pour identifier des stratégies d'élimination, notamment des machines ne s'arrêtant pas.

> Ce que Kopp, Brady, Radó et Lin firent pour l'essentiel, fut d'abord d'imprimer le comportement de certaines des machines retenues, de les étudier et d'essayer de

199. (De Mol 2012, p. 15).
200. (ibid., p. 2).

> détecter certains schémas qui pourraient ensuite être généralisés et prouvés comme étant des cas de boucles infinies. Des programmes furent ensuite écrits pour permettre la détection automatisée des boucles infinies, ce qui permit d'éliminer les machines dont le comportement final correspondait à l'une des boucles infinies trouvées et formalisées dans un programme. Au final, plusieurs types de boucles infinies furent détectés. Les plus importants sont les « boucles simples », les « arbres de Noël », les « arbres de Noël fantômes » et les « compteurs »[201].

L'intérêt de cette démarche, relativement à celle du théorème des quatre couleurs, est qu'elle utilise les ratiocinations automatisées de manière heuristique, c'est-à-dire non seulement pour tester ou prouver des hypothèses, mais également pour en générer. On voit ici la machine jouer un rôle exploratoire dans l'investigation, une exploration qui porte sur l'espace de problème lui-même – dont la taille et la topologie n'ont plus rien de la concision et de l'élégance des théories classiques – arithmétique, géométrie, etc. Une métaphore peut être ici utile pour illustrer cette fonction des ordinateurs dans ce genre d'investigations : car, de même que les machines nous permettent de voler ou de marcher sur la Lune, elles nous permettent à présent d'explorer des espaces intrinsèquement hostiles à l'intelligence humaine, où ne règne aucune des conditions habituelles de ses écologies cognitives. Il suffit d'avoir bien compris quelles étaient les règles de déplacement dans ces espaces pour concevoir des machines capables de les parcourir et d'y accomplir les tâches nécessaires jusqu'à épuisement de leurs ressources. Ce dernier point est important. Cette automatisation ne veut pas dire qu'il n'y a plus besoin de réfléchir, au contraire, puisque le parcours d'espaces systématiques peut se faire sans résultat probant à un coût exorbitant (en temps, en mémoire), mais la réflexion change de nature ; elle consiste à présent à piloter la machine dans des espaces d'inférences organisés systématiquement. Elle devient algorithmique et heuristique, comme nous l'avons indiqué[202].

Outre l'exécution de ratiocinations, les ordinateurs peuvent jouer un second rôle en mathématiques, en quelque sorte symétrique du premier, celui de la vérification de preuves. Il s'agit alors pour eux de confirmer leur *exactitude*, en vérifiant que chaque inférence, aussi minime soit-elle, est autorisée par une règle explicite de la théorie. Ici il est possible de prendre pour (modeste) exemple notre propre preuve des deux premiers théorèmes des *Éléments*, consignée dans l'assis-

201. (De Mol 2012, p. 9).
202. Voir plus haut, section 11.2.

tant de preuve Coq[203]. La preuve se déroule de manière interactive, le mathématicien proposant des actions (dites *tactiques*) à l'ordinateur. Celui-ci vérifie que cette action est autorisée par le système de règles prédéfini (la théorie), et identifie, le cas échéant, quelles hypothèses doivent être vérifiées afin qu'elle soit valide. Le contexte de la preuve est transformé en conséquence : de quels objets et hypothèses le mathématicien dispose, et ce qu'il lui reste à prouver. Une preuve est déclarée valide quand toutes les objectifs de preuve sont atteints. Par exemple, dans l'exemple que nous développons en annexe, on voit qu'il est nécessaire d'expliquer à l'assistant de preuve que l'intersection de deux cercles se situe bien sur chacun d'entre eux, ce qui n'est rien d'autre que *définir* ce que veut dire *intersection* ; cette définition est néanmoins complexe, car selon les cas, elle peut dénoter deux points, un seul, ou aucun.

L'assistant de preuve peut donc aussi être utilisé à rebours, pour vérifier la complétude d'une théorie relativement aux preuves qu'on souhaite y exprimer, et ainsi participer à sa construction. Dans notre réflexion sur la géométrie, nous avons utilisé le programme namingCoq pour nous aider à *clarifier* nos intuitions et à les expliciter de manière distincte. L'assistant de preuve permet ainsi de devenir le chantier de l'élaboration théorique, immédiatement vérifiable sur des théorèmes qui lui servent de test. Il y a ainsi de nombreux projets de transcription de corpus mathématiques dans des assistants de preuves, les uns pour des motifs pédagogiques[204], les autres pour des motifs de recherche[205]. En 2021, l'assistant de preuve Lean permit ainsi de vérifier une preuve ayant un rôle critique dans l'unification de plusieurs théories mathématiques[206]. On peut avoir une idée de la manière dont le travail mathématique pourrait évoluer dans les années à venir par cette remarque d'une mathématicienne :

> Riehl compte parmi les mathématiciens qui ont expérimenté les assistants de preuve ; elle les enseigne même dans certaines de ses classes de premier cycle universitaire. Elle affirme que, bien qu'elle ne les utilise pas systématiquement dans ses recherches, ils ont commencé à modifier la façon dont elle envisage les pratiques de construction de concepts mathématiques, d'énonciation et de démonstration de théorèmes à leur sujet. « Auparavant, je considérais la démonstration et la construction comme deux

203. Voir annexe B.
204. Voir par exemple (Buzzard 2020).
205. Voir par exemple (Voevodsky 2015).
206. (Castelvecchi 2021).

choses différentes, alors que maintenant je les considère comme identiques »[207].

Il nous faut enfin ici mentionner, très rapidement, la technique de la *vérification de modèle* (*model-checking*) qui prend son origine dans la vérification de logiciel, mais qui est mathématique par nature[208]. Complémentaire aux méthodes formelles*, elle tente de vérifier par la « force brute[209] » qu'un modèle formel possède une propriété souhaitée (par exemple, qu'il est cohérent) en explorant de manière systématique une portion finie de l'espace sous-jacent au modèle. Une telle vérification cherche en quelque sorte un contre-exemple. Elle n'est donc capable que de démonstrations négatives à proprement parler et seulement de confirmations provisoires de la conjecture si l'espace engendré est infini, puisque le vérificateur n'en aura exploré qu'une portion finie. Cependant une telle limitation peut être acceptable pour certains problèmes[210]. Ainsi, parmi les langages de spécification mentionnés plus haut[211], Alloy se présente comme un vérificateur de modèle capable de tester les propriétés de toute spécification de relations logiques entre des termes – par exemple, dans le problème des clés des chambres l'hôtel, que nulle intrusion n'est possible.

La vérification de modèle n'est pas assez puissante pour tester des théories mathématiques proprement dites et doit se contenter des « théories modestes » qu'on élabore typiquement pour résoudre des classes de problèmes assez restreintes, c'est-à-dire des modèles[212]. Il est intéressant de remarquer à ce titre qu'elle se développe dans de nombreuses sciences computationnelles, notamment dans les sciences de la vie pour vérifier que le modèle d'un phénomène complexe arbore les propriétés visées – par exemple que tel événement n'arrivera jamais (sûreté*), ou que tel autre finira bien par arriver (vivacité*)[213]. On verra également qu'elle fait son apparition en sciences de l'ingénieur[214].

Ainsi donc, les assistants de preuve et la vérification de modèle sont des outils qui soutiennent l'exploration théorique, c'est-à-dire l'investigation globale ; ils ne servent pas seulement à écrire des preuves systématiques, mais à explorer et à construire le système lui-même. Il est intéressant à ce titre de noter que le para-

207. (Castelvecchi 2021, p. 19).
208. (Baier et Katoen 2008).
209. Pour l'explicitation du sens de cette expression en informatique voir plus haut, § 191.
210. (D. Jackson 2006, p. 3).
211. Voir plus haut, § 201.
212. Sur la différence entre *modèle* et *théorie*, voir plus haut, § 181.
213. (Angius 2017, 2019).
214. Voir plus loin, § 217.

digme de programmation naturel dans ces explorations théoriques est celui de la programmation par contraintes*, qui généralise la programmation logique*, plutôt que celui de la programmation impérative*. Un tel type de programmation ne se présente pas comme la résolution de problèmes particuliers, mais comme la préparation de systèmes pour la résolution de problèmes quelconques – un thème que nous reprenons de manière abondante au chapitre suivant.

§ 216. Les simulations scientifiques

Les transformations qui ont lieu en mathématiques ont également lieu en sciences. Nous abordons ici le seul sujet des simulations, bien connu de l'épistémologie des sciences computationnelles[215].

Les simulations numériques sont aujourd'hui un des modes de représentations principaux des mécanismes[216]. Selon Bechtel et Abrahamsen (2010), elles jouent un rôle essentiel dans l'explication mécaniste, en ce que celle-ci ne peut se contenter de *décomposer* les mécanismes en parties et en opérations élémentaires, mais qu'elle doit également les *recomposer* dynamiquement, afin de montrer comment leur « orchestration en temps réel[217] » génère les propriétés observées. Pour cela, affirment-ils, en s'appuyant sur l'exemple des rythmes circadiens en biologie, la modélisation numérique est un outil indispensable dès que le mécanisme exhibe une certaine complexité. Il permet de localiser quelles parties du mécanisme exactement sont génératrices du phénomène, de l'étudier dans une grande variété de conditions d'environnement, de montrer quelles altérations du mécanisme provoquent des dysfonctionnements, quel est son comportement lorsqu'il est couplé avec d'autres mécanismes, etc. On peut de cela tirer deux observations.

D'une part, si la simulation numérique s'avère être ainsi un complément si naturel à l'analyse du mécanisme, c'est que les équations dynamiques qui le décrivent sont d'emblée procédurales, qu'elles ne visent qu'à nous permettre de « recomposer » le mouvement, comme le dit Bechtel. Les simulations ne sont rien d'autre que des ratiocinations automatisées. Mais là aussi, quelque chose de nouveau se produit, qui outrepasse le simple gain de productivité dans l'investigation. En effet, la complexité de l'analyse des mécanismes est que ceux-ci sont rarement composés

215. Des références concernant ce débat se trouvent dans (Beisbart 2018 ; Symons et Alvarado 2019 ; Varenne 2012 ; Winsberg 2015).
216. (Craver et Darden 2013, ch. 3).
217. (Bechtel et Abrahamsen 2010, p. 3).

des trajectoires simples de systèmes isolés, mais qu'ils décrivent des interactions multiples et parallèles sur plusieurs niveaux.

Ce second point est capital. La valeur de la machine est ici de pouvoir conduire des ratiocinations concurrentes et interactives, c'est-à-dire d'explorer des espaces systématiques composés de systèmes entremêlés. Nous revenons sur le phénomène fondamental de l'interactivité au chapitre suivant. Nous avons ailleurs développé[218] la thèse que c'est *ce fait structurel* de la concurrence des ratiocinations dont est capable la machine qui est la raison centrale de leur opacité épistémique au regard de notre capacité humaine de compréhension, et non par exemple leur nombre, ou leur caractère chaotique. Des procédures initialement écrites pour qu'un investigateur humain puisse les exécuter, ou au moins les suivre, lui deviennent illisibles lorsqu'elles dépendent d'interactions avec d'autres procédures, qui à leur tour dépendent d'autres, etc. Il faudrait, pour reproduire ces simulations par des personnes, disposer d'autant de calculateurs qu'il y a de processus interactifs et, à chaque cycle de simulation, faire exécuter à chacun son segment de ratiocination, puis orchestrer l'échange de leurs résultats pour chaque interaction prévue. Il est par ailleurs possible que, suite à cet échange, certains processus disparaissent, d'autres apparaissent, d'autres soient reconfigurés, etc.[219]

Il n'est pas question ici seulement de systèmes dits *multi-agents*, ces simulations qui étudient les phénomènes statistiques et émergents résultant de l'interaction d'agents dont les règles de comportement sont fixées. La concurrence des ratiocinations peut se faire à un niveau plus élevé d'abstraction, lorsqu'il s'agit par exemple d'étudier l'interaction de deux modèles, qui peuvent porter chacun sur des grandeurs physiques différentes (ainsi un modèle des courants océanographiques et un modèle de la fonte des pôles), ou sur des mêmes phénomènes mais à des échelles différentes (ainsi en écologie, un modèle de comportement statistique de populations et un modèle multi-agents)[220].

Les sciences sont ainsi amenées à devenir computationnelles car, ainsi que l'avance le philosophe des sciences Cliff Hooker[221], l'étude des phénomènes dynamiques dans des conditions stables d'environnement – ainsi la rotation du système solaire, la croissance des plantes, le fonctionnement de l'appareil digestif, etc. – est à peu près achevée. La plupart des problèmes « intéressants » concernent aujour-

218. (Stephanou 2018).
219. Voir plus loin, § 221.
220. (Varenne 2009).
221. (Hooker et al. 2011, p. 9-20).

d'hui des interactions entre phénomènes – qu'il s'agisse de mécanique, de biologie, de neurologie, d'écologie, etc., qui sont rarement linéaires et qui donnent lieu à ce qu'on appelle aujourd'hui, à la suite de Bertalanffy, les *systèmes complexes*.

Ainsi il nous semble peu pertinent de comparer les simulations aux expérimentations, comme on l'a souvent fait. Elles peuvent certes avoir la même fonction, d'aider à valider ou invalider des hypothèses scientifiques en reproduisant numériquement la configuration d'une expérimentation possible[222]. Mais cela se fait sous la forme de ratiocinations, c'est-à-dire par la simple évaluation de systèmes de règles, et non par l'appel à l'expérience, ce qui en font des items radicalement distincts.

Dans les simulations en effet, l'assertabilité garantie d'un fait reste conditionnelle à un système de significations sous-jacent dont on suppose la validité. Par exemple, les simulations dites de *dynamiques moléculaires* modélisent le comportement de la matière au niveau atomique afin d'étudier des propriétés plus macroscopiques, par exemple les effets aérodynamiques de gaz à faible densité sur des aéronefs à haute altitude où les équations d'équilibre classique des gaz ne s'appliquent pas[223].

Il est important de noter qu'on reste ici sur le pur plan des significations, qui permet d'inférer d'un premier jugement (la validité du système de significations représenté par le modèle de dynamiques moléculaires) un second (concernant le comportement attendu et mesurable des gaz autour de l'aéronef) – et qu'il n'y a rien là qui puisse remplacer l'expérience, qui dispose d'une évidence différente et irréductible à celle des inférences entre significations[224]. Il n'est pas question ici de robustesse du modèle ou de fiabilité de l'informatique, dont dépendrait la simulation relativement à l'expérimentation. Cette dernière, de plus en plus, dépend également d'hypothèses et de modèles sophistiqués, d'outils de mesure électronique, et de traitements informatiques qui s'intercalent entre l'expérimentateur humain et son objet. La différence essentielle est que l'expérimentation doit comporter, de manière ultime, une *expérience* sur la chose visée par l'inférence en jeu. Comme le dit Martin-Löf, lorsqu'on veut savoir si le soleil brille il faut, à un moment donné, d'une manière ou d'une autre – qui peut être très sophistiquée –, ouvrir la fenêtre

222. (Beisbart 2018).
223. Pour une introduction à la simulation dite de *dynamiques moléculaires* et ses applications, voir (Boyd et Schwartzentruber 2017).
224. Voir plus haut, § 170.

et regarder le ciel[225].

Ce n'est donc pas vraiment parce qu'elles nous permettent d'économiser des expérimentations que les simulations sont utiles. Leur rôle est d'*étendre le domaine de la systématicité*, c'est-à-dire d'augmenter notre capacité à réaliser des inférences contrôlées sur des objets complexes de l'expérience (aux interactions non linéaires), là où les possibilités de simplification de la ratiocination sont épuisées. C'est par ce biais seulement qu'elles peuvent remplacer des expérimentations – celles qui étaient réalisées, non pour valider une théorie, mais pour tenter d'éclairer, tant bien que mal, des phénomènes dont on peut bien avoir la théorie, mais pas une théorie *utilisable*. Ainsi, les essais de soufflerie en aéronautique ont diminué radicalement depuis l'avènement des simulations de dynamique des fluides, sans pour autant disparaître[226].

Cette perspective explique à la fois pourquoi il n'y a en effet rien de neuf dans les simulations[227], car elles ne sont qu'une ratiocination, que les scientifiques et les techniciens ont toujours cherché à déléguer pour se focaliser sur l'investigation et la formation d'hypothèses – et à la fois pourquoi elles sont radicalement neuves, car elles permettent de tester l'interactivité des règles entre elles, ce que ne peut faire une ratiocination classique – et ainsi d'étendre le champ de la systématicité scientifique aux objets complexes de l'expérience, ce qu'on appelle aujourd'hui la science des systèmes complexes[228].

§ 217. La conception assistée par ordinateur

L'informatique de conception (ou CAO*) s'est développée progressivement, d'un simple outil à dessiner des plans et des perspectives qu'elle était, jusqu'à devenir un environnement complet de développement, s'enrichissant de toutes les propriétés utiles de la programmation. Nous suivons en partie ici la thèse récente de Gaudillière-Jami (2022), qui retrace de manière très complète l'histoire de l'informatique architecturale, pour montrer en quoi l'interactivité entre l'homme et la machine augmente la puissance de l'investigation. Il convient de noter que notre développement ne prend pas position sur le débat concernant la possibilité et la pertinence de réduire l'essentiel du travail architectural et d'ingénierie à des

225. Voir plus haut, § 163.
226. (Stumpe 2018).
227. (Frigg et Reiss 2009).
228. (Hooker et al. 2011).

procédures ou à des algorithmes. Nous nous intéressons seulement aux possibilités que l'informatique offre à l'architecte et à l'ingénieur de faire des choses plus efficacement, ou des choses nouvelles – sans supposer que ces choses-là constituent l'essentiel de leur travail. Notamment, nous ne disons rien ici de l'interaction entre la créativité de l'architecte et les possibilités et contraintes de ces nouveaux outils, question sur laquelle se penche Gaudillière-Jami.

La CAO*, quand elle a commencé à être effectivement utilisée par les praticiens, n'était rien d'autre qu'un outil de dessin, qui permettait de produire des plans et des images numériques. Les données entrées par l'architecte avaient néanmoins déjà des propriétés dynamiques, puisqu'elles permettaient, dès l'origine, de générer de multiples vues et coupes de l'objet, en trois dimensions. Ces simulations graphiques donnaient un retour immédiat à l'ingénieur sur les conséquences de ses actions – par exemple que telle colonne obturerait la vue du paysage depuis telle fenêtre.

Une seconde étape fonctionnelle, qui marque le début d'une modélisation profonde, appelée BIM* (*Building Information Modeling*), est la structuration d'un projet en calques qui permettent de visualiser séparément ou simultanément plusieurs propriétés du bâtiment : son apparence externe finale, la structure brute dépouillée des revêtements et embellissements, les réseaux d'eau, d'électricité, de chaleur, etc. Cette superposition permet de laisser des équipes spécialisées (structure, plomberie, électricité, etc.) travailler séparément tout en vérifiant régulièrement la compatibilité de leurs conceptions (par exemple un câble ne peut traverser un tuyau, ou un interrupteur ne doit pas être placée derrière un radiateur). Nous disons que cette modélisation est profonde non seulement en raison de la superposition de ces multiples plans, mais parce que les données y deviennent actives. On parle d'ailleurs dans le métier de « conflits de pixels » pour mentionner les superpositions impossibles ou défavorables, ce qui montre bien leur nature profonde de *règles*.

Les nouveaux systèmes BIM permettent à présent d'effectuer automatiquement certaines de ces vérifications, car ils « comprennent » les règles de superposition applicables, qu'elles relèvent de la physique pure et simple, de la convenance pratique, voire de l'esthétique. Ce sont notamment la vérification des normes réglementaires (sécurité, environnement, urbanisme) qui sont actuellement la priorité. On fait pour cela appel à la technique de la *vérification de modèle* déjà évoquée[229].

229. Voir plus haut, § 215.

Andrich et al. (2022) donnent une vision récente de l'état de l'art sur la question et note :

> Cette possibilité ouvre la voie à un important changement de mentalité. Si dans un processus traditionnel, en effet, tout contrôle ne peut être effectué que sur une base d'échantillons, dans un processus basé sur le BIM, la possibilité d'une mise en œuvre automatique ou semi-automatique des contrôles permet une analyse beaucoup plus approfondie. À cet égard, il convient de considérer que « dans les processus de conception standard, seuls 5 à 10% du contenu informatif du projet sont systématiquement vérifiés. La vérification des modèles conduit à une validation automatisée de 40 à 60% de la conception, en suivant des contrôles spécifiques plutôt que des contrôles par échantillonnage ». La possibilité de créer et de réutiliser, au sein du logiciel de la vérification de modèle, une base de données de règles testées et fiables permet d'ajouter la caractéristique de répétabilité et d'objectivité aux contrôles à effectuer, garantissant ainsi la standardisation du processus, avec pour conséquence la réduction de la marge d'erreur et le raccourcissement du temps nécessaire pour terminer le processus de contrôle[230].

D'autres processus de vérification ne s'appuient pas sur des règles géométriques, mais sur la simulation de certaines dynamiques de l'objet conçu. Ainsi, toujours en génie civil, il s'agit de soumettre un modèle de bâtiment ou d'un ouvrage d'art à des simulations qui testent sa réaction à différentes tensions (poids, vents, tremblements de terre, chocs, etc.), calculent sa performance thermique et énergétique, ses propriétés acoustiques, la luminosité des pièces à toute heure et en toute saison, etc.[231] La figure 11.5 donne l'exemple d'une simulation énergétique obtenue grâce au BIM. Ces simulations sont possibles parce que les outils de CAO actuels ne manipulent pas seulement des formes et textures de murs, portes et fenêtres, mais des objets techniques, concrètement des fichiers de données BIM, établis par les fabricants de matériaux et des pièces de construction, qui déclarent aux logiciels de CAO toutes leurs propriétés : forme géométrique et apparence, mais également résistance aux chocs, transparence, conductivité thermique, etc. qui les rendent disponibles pour n'importe quelle simulation. Ils se comportent, dans un projet en cours de conception, comme des objets actifs qui *interagissent* entre eux et avec le système global du bâtiment.

Du moment que ces règles sont explicitées, la modélisation sous contraintes

230. (Andrich et al. 2022, p. 5).

231. Utkucu et Sözer (2020) donne un état des lieux récents de ces méthodes encore en voie de maturation.

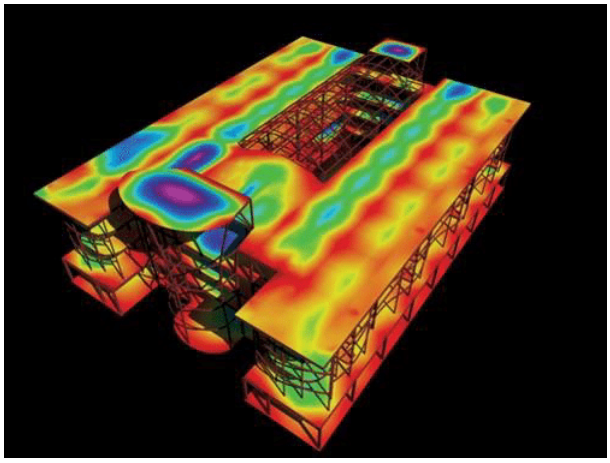


FIGURE 11.5 – Simulation énergétique d'un bâtiment obtenue grâce au BIM
Source : NICAŁ et WODYŃSKI (2016)

peut être en partie automatisée. Les logiciels de CAO peuvent aujourd'hui générer automatiquement des structures qui répondent au cahier des charges établi par l'architecte. Celui-ci peut alors sélectionner une ou plusieurs propositions, et les modifier directement, ou poser de nouveaux problèmes d'optimisation à la machine. Ces outils sont aujourd'hui le plus souvent utilisés dans le cadre de problèmes fonctionnels, comme optimiser l'usage d'un espace de bureaux en maximisant, par exemple, la luminosité et l'isolation sonore des postes de travail, mais leur périmètre d'application s'étend progressivement. Ici, non seulement le système de conception dispose d'un modèle des propriétés profondes de l'objet étudié, mais il est également capable d'automatiser une grande partie des tâches de l'investigation en explorant systématiquement des espaces de possibilité comme nous l'avons vu dans les exemples mathématiques ci-dessus. La tâche de l'architecte ou de l'ingénieur est fondamentalement modifiée : il ne s'agit plus de trouver la solution, mais seulement de bien poser le problème, qui s'exprime par la définition de la bonne *fonction-objectif*, terme par lequel on désigne le système de contraintes à résoudre.

Enfin, à un niveau encore supérieur, les architectes peuvent utiliser des programmes informatiques pour imaginer des structures génératives, c'est-à-dire résultant de l'application de fonctions récursives complexes, comme les L-systèmes permettant de modéliser la croissance des plantes (voir figure 11.6)[232].

232. Voir (GAUDILLIÈRE-JAMI 2022, p. 276), et plus généralement p. 39-54, 275-315 et 453-474.

FIGURE 11.6 – *The Tote*, par Series Architects –
Source : https://www.serie.co.uk/projects/269/the-tote

À travers ces six niveaux de programmation possibles que nous venons d'évoquer dans le processus de conception : interactivité visuelle simple, vérification collaborative de cohérence, vérification automatisée, simulation dynamique, génération automatique de plans, structures génératives, on voit que la conception est transformée par la programmation dans l'exacte mesure où elle requiert des contrôles de cohérence d'hypothèses avec des systèmes de règles, et où elle cherche à concevoir ses objets par l'exploration systématique des espaces de possibilité ouvertes par ces systèmes.

Comme nous l'avions noté au début de notre réflexion, la programmation, dans ces contextes, n'apparaît plus comme *programmation*. L'utilisation avancée des fonctionnalités de CAO décrites peuvent s'apparenter à du code[233], mais l'architecte niera qu'il est en train de programmer : il est en train de concevoir son modèle et d'éprouver sa robustesse relativement aux contraintes et aux objectifs qu'il s'est fixés.

233. (GAUDILLIÈRE-JAMI 2022), en particulier section 4.2 : la programmation heuristique.

§ 218. L'investigation systématique

Il nous faut à présent interpréter ces résultats et présenter une synthèse générale de notre réflexion sur la nature et le rôle de programmation au sein de la connaissance rationnelle.

La machine nous est apparue, au cours de ce chapitre, comme le lieu idoine d'exécution des procédures exactes dans lesquelles se récapitulent des investigations. *Exactitude* veut dire ici que la ratiocination procède entièrement selon des règles prédéfinies, sans appel au jugement. Il n'est aucune inférence qu'elle réalise dont on ne puisse citer la règle qui l'a motivée. La machine est le lieu idoine de projection de telles ratiocinations parce qu'elle désigne un phénomène de l'expérience dont les règles de comportement peuvent être décrites dans un système simple, car elle a un environnement *pauvre* dont l'influence peut être aisément identifiée et contrôlée. Elle est un lieu d'exactitude car il n'y a pas de mauvaise surprise possible (idéalement) dans l'explication ratiocinative de son comportement.

Du moment que cet environnement est néanmoins suffisamment riche pour que tout autre système de significations cohérent puisse y être codé – comme par exemple dans les contrôles de transmission ayant la puissance de machines de Turing – elle peut représenter n'importe quelle ratiocination exacte. Il est ainsi possible de garantir qu'elle restera *sourde* à toute inférence qui ne serait pas issue des règles élémentaires du système.

Les investigations systématiques visent à se récapituler dans des procédures exactes qui parcourent exhaustivement leurs conditions, ce qui requiert qu'elles soient exprimées dans les systèmes de significations contrôlés et délimités que sont les théories. On peut donc affirmer que *les machines y sont chez elles*, et que c'est de là que provient l'injonction des logiciens post-hilbertiens de *procéder mécaniquement*. Nous pouvons enfin comprendre pourquoi cette métaphore est si puissante : c'est que les machines, du fait de la simplicité conçue de leur environnement, sont le lieu par excellence où peut être tenté le passage de l'approximation à l'exactitude. Ces machines sont l'image matérielle d'une ratiocination exacte qui rend tout recours au jugement inutile.

Aussi est-il naturel de les voir s'insérer dans les investigations systématiques, qu'elles soient locales (résolution de problèmes) ou globales (réflexion et construction théorique). Dans le premier cas, ce sont en effet les machines qui peuvent le mieux parcourir, selon des règles inductives, ces espaces dans lesquels les règles de la ratiocination doivent être exprimées. Elles servent bien sûr à réaliser des gains de

productivité (ainsi la génération automatique de visualisations à partir d'un plan 3D en CAO*) mais surtout à construire des solutions pratiquement inabordables aux personnes humaines – ainsi le théorème de quatre couleurs, les simulations de dynamique moléculaire, ou les structures génératives des architectes. Elles peuvent également permettre d'explorer un espace de systématicité afin d'engendrer des hypothèses et des heuristiques – ainsi l'usage de l'ordinateur dans le problème des castors affairés ou la génération automatique de plans en architecture relativement à des contraintes prédéfinies. L'investigation se déroule alors au niveau supérieur relativement aux ratiocinations, en ce qu'elle ne se préoccupe plus de les exécuter elle-même, mais de les organiser stratégiquement afin de parvenir à ses fins – elle supervise des machines assignées au parcours de vastes espaces de règles qui, pour reprendre la métaphore du voyage spatial, sont des « milieux hostiles » à l'intelligence humaine et à ses écologies cognitives.

Les machines n'ont cependant pas seulement cette fonction dans la résolution systématique de problème. Elles peuvent également servir à vérifier l'exactitude d'une solution, c'est-à-dire sa conformité avec un système de règles préétabli. C'est le rôle des assistants de preuve en mathématiques, ainsi que des techniques de superposition de plans, de vérification de modèle et de simulation dynamique en ingénierie civile grâce au BIM*.

Comme nous l'avons vu, cette fonction de vérification peut servir également en sens inverse, à tester une construction théorique relativement aux exigences d'un problème connu, notamment *via* les assistants de preuve en mathématiques. Mais les simulations peuvent bien sûr également jouer ce rôle, lorsqu'il s'agit d'explorer ou de tester une hypothèse scientifique, ainsi qu'on l'a vu dans leur rôle d'élucidation des mécanismes en sciences de la vie. Ainsi dans l'investigation théorique (globale), ce sont également les machines qui peuvent le mieux l'aider à explorer la pertinence d'hypothèses, de définitions ou de connexions entre propositions.

Dans cette perspective, il devient enfin possible de caractériser la programmation comme entreprise de connaissance relativement aux mathématiques, aux sciences et à l'ingénierie – une question qui nous suit depuis le début de notre réflexion[234]. Si les mathématiques ont affaire à l'invention de systèmes procéduraux purs qui peuvent être mobilisés par les sciences et l'ingénierie dans la construction de leurs théories et la résolution de leurs problèmes, la programmation, quant à elle, consiste à mobiliser des machines pour rendre effectifs ces moyens procé-

234. Voir plus haut, § 12.

duraux au sein de l'investigation. Ce faisant, la programmation a à la fois son domaine propre – puisqu'elle a affaire à des machines – et à la fois elle s'infiltre totalement au sein des mathématiques, des sciences et de l'ingénierie dont elle sert les desseins.

Si cela est possible, c'est que celles-ci, *dès le départ* (c'est-à-dire dès les Temps modernes), visent l'établissement et l'exploitation de systèmes de significations contrôlées dans lesquels toute investigation réussie pourrait en droit être décrite par une procédure qui s'agrégerait au système et deviendrait ainsi disponible comme outil d'investigations futures, sous la forme d'une pure ratiocination. En automatisant cette dernière, les machines permettent d'étendre la complexité des règles praticables, tout en gardant les ratiocinations sous contrôle d'exactitude ; ce faisant elles augmentent également le champ possible de la construction théorique utile. L'investigation systématique se trouve ainsi dotée en surcroît d'une capacité réflexive sur elle-même, puisqu'elle peut s'occuper à présent d'agencer les ratiocinations plutôt que les exécuter. Il convient de préciser ces deux propriétés.

Concernant l'extension de la complexité des règles, on peut remarquer que les schémas inductifs de systématicité permettent d'engendrer des règles *au moment venu*, en fonction des paramètres que le schéma de règles trouve en entrée. Certes, de telles règles paramétriques ont toujours existé en mathématiques, mais l'automatisation permet de rendre ces schémas inductifs plus complexes à souhait, contrairement à l'intelligence humaine, qui est pratiquement limitée dans les manipulations de règles[235]. Les règles deviennent ainsi dynamiques et *sensibles* au moment précis de l'investigation où elles sont sollicitées et à l'état de transformation du problème, comme l'évoque Knuth[236].

Concernant la capacité de réflexivité dont se dote l'investigation grâce aux machines – et qui lui permet, comme nous l'avons vu, de vérifier l'exactitude de son propre processus de pensée, d'identifier des heuristiques et des hypothèses nouvelles, d'élaborer des stratégies de ratiocination – il faut en préciser la singularité. La réflexivité est une propriété élémentaire de l'investigation, qui ne cesse de se récapituler elle-même, selon plusieurs guises : dans l'action, elle fait du geste découvert ou appris un geste acquis, une habitude, qui devient disponible pour

235. Voir plus haut, p. 317.
236. Voir plus haut, p. 761. Sur la notion de règle dynamique, voir plus loin, § 224.

d'autres gestes plus complexes[237]. Dans le jugement, elle élabore de nouvelles significations à partir des significations disponibles, et vient ainsi enrichir la capacité à exprimer des idées[238]. En mathématiques, les objets procéduraux se construisent et se composent les uns à partir des autres[239]. C'est une autre forme de réflexivité encore que rendent possibles les machines, et elle concerne l'exécution de règles – la ratiocination. Nous avons très tôt noté la propriété des machines d'être des objets récursifs – de pouvoir se produire les unes les autres, ce qui a pris un sens très concret lorsque nous avons étudié les machines universelles[240]. C'est donc le *procès* même de l'investigation qui, par les machines, peut se prendre pour objet, se composer et itérer sur lui-même à volonté. Grâce à elles, l'investigation a ici affaire *essentiellement* à la classe (C) des impératifs que Simon avait identifiés à propos de la résolution de problème, ceux qui lui intiment de trouver les *méthodes de construction* des procédures, plutôt que les procédures elles-mêmes[241].

La forme systématique est requise pour que l'investigation puisse accéder, de manière stable, et à grande échelle, à cette forme de réflexivité. On voit bien en effet qu'on ne saurait aller très loin dans la manipulation de procédures approximatives ou sensibles aux exceptions. Développer des investigations qui manipulent d'autres investigations requiert que celles-ci prennent la forme de procédures exactes – sans surprise quant à la signification de leurs prescriptions – et exhaustives – sans surprise quant aux conditions rencontrées dans l'investigation. Les machines sont également *pratiquement* requises afin que cette réflexivité puisse s'appuyer sur des expériences sensibles : il faut qu'on puisse *observer* l'exécution ratiocinative des procédures ainsi composées dynamiquement afin de pouvoir les corriger ; la fréquence des bogues* en programmation, qui laissa les premiers informaticiens perplexes[242], est le signe de ce niveau supérieur de réflexivité qui est requis de l'investigation, et qui pratiquement ne peut être développé sans cette *traçabilité* de la ratiocination qu'offre l'informatique.

Nous pouvons à présent revenir à la vision d'Herbert Simon qui voyait dans la programmation le modèle de la résolution de problème. Celle-ci touche juste, lorsqu'elle voit la programmation des machines comme la forme exacte de la réso-

237. Voir plus haut, § 168.
238. Voir plus haut, § 173.
239. Voir plus haut, § 176.
240. Voir plus haut § 119, § 185
241. Voir plus haut, § 65.
242. Voir plus haut, § 10.

lution de problème procédant par la seule manipulation de règles dans un espace lui-même parcouru selon des règles, *i.e.* comme un problème de recherche (*search problem*)[243]. Cependant, nous corrigeons cette thèse de manière nette en reconnaissant qu'il s'agit là d'une forme très spécifique de l'investigation. Ici, ce sont les thèses cognitives de Simon, qui voulaient que le cerveau calcule, qui l'ont poussé à étendre indûment son intuition. Il ne faut pas dire que nous raisonnons comme des machines, car cela serait tout simplement faux, mais que, dans une démarche systématique de résolution de problème, nous cherchons à raisonner aussi bien que lorsque nous raisonnons à leur propos ; et ce que la programmation offre c'est, en tout problème qu'on vise à traiter systématiquement, *de raisonner avec elles*.

Par ailleurs, elle ne doit pas être vue comme une « science de l'artificiel » qui aurait des méthodes complémentaires à celles des sciences naturelles – une distinction qui posa de nombreux problèmes à Simon puisqu'il voyait bien que la programmation avait aussi un rôle à jouer dans toute investigation scientifique. La programmation a trait aux résolutions de problèmes et s'occupe donc d'investigations locales ; mais comme toute investigation théorique se réalise et s'éprouve dans les problèmes qui lui sont adressés, la programmation peut par ce biais y contribuer, de la manière que nous avons indiquée.

Il faut donc renverser la perspective concernant la transformation que connaîtrait la connaissance rationnelle avec le développement des sciences computationnelles. S'il y a transformation, elle nous semble aller dans le sens de l'accomplissement de son projet systématique qui, nous le comprenons à présent, peut être caractérisé au moins de trois manières complémentaires :

1. Il vise à établir des systèmes de significations contrôlées pour soutenir l'investigation, ce que nous avons appelé la systématicité.
2. Il est cumulatif, en ce qu'il doit permettre à toute investigation réussie d'être adjointe à ce système comme une signification complémentaire disponible.
3. Il est réflexif, au sens où il doit permettre la composition de ratiocinations pour résoudre des classes toujours plus larges de problèmes et économiser au mieux les investigations inutiles.

Pour en rester à ce troisième point, il est donc partie intégrante de ce projet que se développe *la science automatisée*, si on entend par là les « méthodes scientifiques qui peuvent être mises en œuvre sans l'intervention de personnes, si ce n'est

243. Voir plus haut, § 65.

pour initier et, le cas échéant, entraîner le processus[244] ». Humphreys donne pour exemples l'analyse automatisée des données gigantesques fournies par les télescopes astronomiques et les accélérateurs de particules. Il s'agit certes là de tâches purement ratiocinatives qu'on pourrait qualifier de subalternes dans la découverte scientifique. L'exemple du célèbre AlphaFold, sur lequel nous revenons plus loin[245], est plus parlant. Bien qu'on soit encore très loin de l'automatisation complète du problème fondamental de biologie moléculaire auquel ce logiciel d'apprentissage profond* a contribué, il indique une direction certaine de la recherche scientifique, à savoir qu'une classe de problèmes bien posée peut être en droit résolue par les machines entièrement. L'investigation, dans ces cas de figure, consiste principalement à trouver des algorithmes et des heuristiques, ou encore des reformulations du problème qui augmentent la productivité de la machine et qui établissent la correction de son système d'inférence. Mais ces tâches à leur tour peuvent également être partiellement automatisées.

Le point essentiel, comme nous l'avons déjà dit, est que la machine peut également jouer un rôle dans la clarification du problème lui-même et de ses bonnes méthodes d'analyse – et c'est exactement ce qu'on observe aujourd'hui de l'usage d'AlphaFold : il est une source de créativité méthodologique pour les scientifiques[246]. La machine joue ici le rôle d'un partenaire, comme on l'a vu dans la plupart des exemples ci-dessus, l'enquêteur émettant des hypothèses et la machine lui en indiquant la correction et les conséquences. Une science automatisée ne veut donc pas dire que toute investigation y serait devenue inutile, mais une science d'un niveau supérieur, occupée principalement, plutôt que de résoudre directement des problèmes, de formuler les problèmes intéressants, de composer des stratégies ratiocinatives pour leur résolution, d'inventer de nouvelles structures de systématicité, d'étendre et de maintenir les infrastructures théoriques de l'investigation.

Dans cette transformation il n'est plus possible de voir l'automatisation de l'investigation scientifique sous le prisme unique de simples gains de temps réalisés sur les ratiocinations, ces étapes ennuyeuses et inévitables du voyage. Dans le projet moderne de la science, l'automatisation est bien plutôt partie de la *destination*.

244. (Humphreys 2020, p. 11).
245. Voir plus loin, § 239.
246. (Callaway 2022).

Chapitre 12

Rendre les règles effectives

§ 219. Introduction

Au chapitre précédent[1], nous avons vu que la notion de *procédure effective* excluait l'interaction de l'exécutant avec son environnement, du fait de sa focalisation sur le modèle classique du calcul. Elle a une forme *fonctionnelle*, où l'ensemble des paramètres du problème sont énumérés tout d'un coup, comme ses variables. Elle est une séquence de règles qui s'appliquent exactement, sans interruption possible, jusqu'à la conclusion de l'inférence.

Or il n'y a que très peu d'investigations de la vie pratique qui peuvent être représentées sous cette forme fonctionnelle idéale. Une investigation, on l'a vu, est d'abord et avant tout un dialogue avec une situation problématique, qui se déroule certes sur le plan des significations, mais qui se confronte sans cesse à l'expérience. Reconstruire cette investigation, même idéalement, exige de représenter aussi, fût-ce de manière partielle, la situation elle-même, dans toute ses variations possibles, et avec toute la profondeur nécessaire à l'intelligibilité des décisions qui seront prises. Un concept plus riche de procédure apparaît ici, qui ne représente pas une simple séquence d'opérations à exécuter pour passer d'une situation problématique à une situation résolue, mais une certaine interaction réglée avec elle.

Une telle procédure est *interactive* comme l'est l'investigation qu'elle récapitule. Il ne faut plus la représenter comme un simple mécanisme de transmission allant d'un point A à un point B, mais plutôt (si on maintient l'image de la transmission) comme un réseau de liaisons entrelacées, dont les multiples extrémités peuvent être sensibles chacune aux signaux de l'environnement, et qui, tel une

1. Voir plus haut, § 193.

toile d'araignée, les répercute vers l'ensemble de ses parties et leur répond en retour. Une telle possibilité se trouve déjà, à l'état latent, dans l'analyse par Marx de la machine industrielle et des propriétés spécifiques du mécanisme de transmission, qui, capable de distribuer une même force à plusieurs machine-outils et de *coordonner leurs opérations*, les rend potentiellement interactives les unes relativement aux autres, de sorte que la machine ne doit plus être comparée à l'opération d'un seul ouvrier, mais à celle d'un atelier tout entier[2]. C'est au développement de cette idée – à bien des égards, seulement esquissée – que ce chapitre est consacré.

Cette possibilité nouvelle qu'apporte la programmation, d'organiser l'*interaction* de procédures, est à l'origine des changements les plus radicaux qu'elle engendre.

Nous commençons par exposer quelques éléments de la théorie informatique de l'interactivité et de la concurrence, qui fut à l'origine de bouleversements pratiques et conceptuels importants de la discipline dans les années 1970, entre autres grâce aux travaux d'Alan Kay, Carl Hewitt, et Robin Milner, qui tous trois proposèrent, sur des plans différents, de refonder l'informatique sur la notion d'*interactivité* (section 12.1).

Ces changements ne sont pas de nature « technique », si on entend par là la transformation de procédés déterminés de production ou de pratiques en général. La programmation de procédures interactives permet de *rendre des règles directement effectives* dans une situation problématique. Par là nous voulons dire qu'une règle sera exécutée automatiquement, sans l'intervention d'une personne humaine, si sa condition se trouve réalisée dans la situation. Cela permet de coupler des objets matériels (comme un portillon d'accès[3]) ou institués (comme un compte bancaire) à des règles et à des procédures et de les rendre ainsi *interactifs* (nous revenons sur le choix de ce terme en fin du développement § 225). C'est en cela que la programmation peut ne plus être vue comme programmation *de machines*, mais comme programmation directe des objets de la situation. Les ordinateurs, d'ailleurs, s'invisibilisent très concrètement en devenant, soit des puces électroniques cachées au sein des mécanismes matériels, soit des machines virtuelles de l'informatique en nuages*.

C'est ici que la programmation augmente, non seulement notre pouvoir d'investigation, mais également notre pouvoir direct d'agir et de transformer le monde,

2. Voir plus haut, § 205.
3. Voir plus haut, § 55.

en nous offrant de concevoir des séquences de règles qui résolvent des problèmes intrinsèquement interactifs, – alors qu'avant elle de telles situations étaient naturellement comprises comme le lieu propre de l'intelligence humaine, celui de son adaptabilité aux circonstances. Autant avions-nous, au chapitre précédent, insisté sur la continuité sur le temps long qu'avait la programmation avec la construction de mécanismes aussi anciens que les montres et les systèmes d'irrigation, autant ici s'agit-il mettre en évidence les possibilités radicalement nouvelles qu'apportent les ordinateurs quant à la résolution de problème (section 12.2).

En particulier, les objets interactifs étant en dialogue avec la situation, cela implique que celle-ci ne présente pas d'irrégularités auxquelles ils ne sauraient répondre ; elle doit se comporter systématiquement. *A minima*, cela signifie qu'elle peut être décrite par un certain espace régulier de conditions[4]. Cependant, une telle description devient rapidement complexe, du moment que plusieurs objets interactifs peuvent interagir au sein d'une même situation, puisque le comportement de l'un peut devenir la condition de l'autre, et vice-versa. C'est donc toute la situation elle-même qui doit devenir interactive, et la programmation porte alors, non plus seulement sur des objets, mais sur des systèmes interactifs. Cela transforme de manière radicale les possibilités des *systèmes objectifs* qui encadrent de règles les pratiques humaines, *i.e.* les institutions et les ensembles techniques[5]. Ces transformations entraînent la prolifération de règles de plus en plus finement adaptées aux circonstances, qui relèguent cependant les personnes humaines à la périphérie des situations problématiques (section 12.3).

On pourrait croire qu'on ne parle là que des utilisateurs de tels systèmes (administrés, usagers, employés, techniciens, etc.) et que les programmeurs se trouvent au contraire en situation d'omniscience et d'omnipotence. Il n'en est rien, car face à des systèmes existants, la programmation devient elle-même un exercice interactif et partiel, qui doit chaque fois négocier l'insertion prudente de nouvelles règles face à une dette et une insécurité réglementaires qui augmentent de manière cumulative (section 12.4).

Ce chapitre est, encore une fois, largement inspiré des travaux de Michael Jackson, et notamment de son ouvrage *Problem Frames*[6]. L'intérêt de sa réflexion est qu'elle se situe d'emblée dans la perspective de la résolution de problème qui est

4. Voir plus haut, § 58.
5. Voir plus haut, section 10.5.
6. (M. Jackson 2001b).

la nôtre. On se rappelle son schéma *machine - situation* sur la base duquel nous avons élaboré notre interprétation contractuelle de la programmation : programmer, c'est résoudre un problème dans une situation à l'aide d'une machine. Au chapitre 3 de son ouvrage, il expose quatre problèmes élémentaires d'interactivité, qu'il se garde bien de présenter comme exhaustifs, mais qui selon lui composent la plupart des problèmes interactifs plus complexes qui nous intéressent. Ces problèmes répondent à des cas de figure simples comme un thermostat, la commande d'un système d'écluse, l'affichage électronique de l'état d'un système physique, l'interaction d'un utilisateur avec un document électronique. Nous ne détaillons pas ces problèmes élémentaires, qui ne sont pas directement utiles à notre propos, mais ensemble ils esquissent selon nous une *logique des problèmes interactifs*.

Par l'examen détaillé de ces problèmes, M. Jackson montre que l'interactivité est une fausse évidence, et que des logiques complexes peuvent être requises pour obtenir le comportement qui nous semble naturel. La machine a toujours le rôle d'un médiateur entre deux domaines distincts de l'environnement, l'un qui couvre les phénomènes d'intérêt, et l'autre qui représente le rôle de contrôle ou de commande, souvent humain.

Une autre manière de voir ces problèmes est qu'il visent le *couplage* d'un processus informatique avec un domaine de l'expérience, de telle sorte que l'opérateur n'ait plus à interagir directement avec ce dernier, mais seulement avec la machine. C'est dans des problèmes aussi simples que celui d'un thermostat ou de la commande d'une écluse que se joue l'ancrage des procédures computationnelles dans le monde de l'expérience.

Notre réflexion requiert d'analyser l'interactivité à un niveau supérieur de composition, là où elle s'assemble en objets et en systèmes. Dans les sections 12.2 et 12.3, nous présentons ainsi quatre schémas courants de programmation interactive, que nous nommons *registre, flux de tâche, espace de travail* et *réseau de services*, qui sont donc globalement inspirés de la réflexion de Michael Jackson. Ils forment la colonne vertébrale de ces deux sections et nous permettent de détailler les principales modalités selon lesquelles adviennent les objets et les systèmes interactifs.

Pour achever cette introduction, il faut noter que nous substituerons souvent aux termes *programme* et *procédure* celui plus générique de *système de règles dynamiques* pour désigner les programmes interactifs. Nous nous en expliquons en § 224. Sur la distinction entre *environnement* et *situation*, ainsi que sur le terme

avantage, nous renvoyons à la note terminologique en fin de § 192.

12.1 La théorie de l'interactivité

Cette section introduit la notion de procédure interactive (§ 220) et montre qu'elle est représentée de manière inadéquate par la théorie classique de la calculabilité (§ 221). Nous présentons les refondations de la programmation proposées par trois informaticiens, Alan Kay, Carl Hewitt, et Robin Milner, qui établirent (avec d'autres) ce constat dans les années 1970-1980 (§ 222).

§ 220. L'interactivité

Nous appelons processus une procédure ou un programme en cours d'exécution. Nous dirons qu'un processus (et par extension, une procédure) est interactif, s'il peut recevoir, au cours de son exécution, une donnée provenant de son environnement. Dans certains cas, il peut *avoir besoin* de cette donnée, et tant que celle-ci n'est pas fournie, le processus est en suspens. De telles situations sont courantes : un programme qui attend que l'utilisateur entre une information sur son clavier, un serveur Web qui attend une connexion d'un visiteur, une machine industrielle qui attend de nouveaux flux de matières premières. Cette interactivité est réalisée soit par l'envoi explicite d'une requête vers un point donné de l'environnement, et l'attente de sa réponse, soit par l'interrogation d'un fichier dynamique, également appelé flux (*stream*), qui peut être modifié par l'environnement. Tant que ce fichier est vide, le processus est bloqué, et il réitère sa tentative de lecture régulièrement. Quand le fichier contient une information dont il a besoin, le processus la récupère et continue son exécution. Ce second mécanisme permet à l'environnement d'envoyer des informations au processus par avance, sans attendre que celui-ci les demande.

Cette interactivité relie, dans une certaine mesure, le temps d'exécution physique du processus au temps de l'environnement, puisque le processus continue son exécution uniquement au moment (de l'environnement) où la donnée lui est fournie, et si ces opérations de lecture sont régulières, son rythme fondamental est d'une certaine manière coordonné avec celui de l'environnement. On peut donc l'appeler interactivité *coordonnée*, ou encore *à rendez-vous*.

Un cas extrême d'interactivité coordonnée est celui où le processus attend sys-

tématiquement une autorisation de l'environnement pour réaliser la moindre tâche élémentaire. Dès qu'il l'a réalisée, il se met en attente d'une prochaine autorisation. Ce mécanisme est utile pour synchroniser de manière forte plusieurs processus entre eux, chacun recevant tour à tour une autorisation d'exécution, de sorte qu'ils avancent tous dans le même temps logique. On appelle souvent cette temporalité *synchronisée*, même si ce terme est utilisé en des sens différents par les ouvrages sur la question.

L'interactivité coordonnée correspond à de nombreux cas de figure concrets, où le processus « pose des questions » à l'environnement et attend ses réponses pour progresser. Il peut s'agir de questions au sens propre, adressées à un interlocuteur, mais également de consultations d'archives, de tests expérimentaux sur des dispositifs physiques, de coups dans un jeu, etc. Parfois l'information demandée n'est pas dépendante de la temporalité de la question (consultations d'archives), parfois elle dépend de son temps physique (par exemple la météo) ou de son temps logique (par exemple le coup de l'adversaire dépend des coups joués jusqu'ici).

Dans l'interactivité coordonnée la procédure décide, dans son temps d'exécution logique, *quand* demander l'information. Par contraste, on pourrait appeler *interactivité non-coordonnée* les cas où l'information *doit* être reçue par la procédure quel que soit l'état dans lequel elle se trouve, et réagir immédiatement en interrompant sa tâche courante pour exécuter une sous-procédure adaptée à l'information reçue, ou au moins décider quoi faire. Les protocoles d'urgence ont cette forme ; les hôpitaux reprogramment de manière continue leurs priorités en fonction du flux d'entrée des patients.

Dans le cas général l'interactivité non coordonnée peut modifier de manière non explicite et surtout indéterminée le fonctionnement de la procédure. Considérons l'exemple donné par MILNER (1993, p. 80), de deux programmes P1 et P2 ayant le code suivant :

```
1    P1: x = 1; x = x + 1.
2    P2: x = 2.
```

Ces deux programmes doivent donner le même résultat (`x == 2`). Cependant si un autre programme Q ayant le code :

```
1    Q: x = 3.
```

tourne en parallèle de P1 ou de P2, la valeur de `x` peut se trouver être finalement 2, 3 ou 4 dans le premier cas, ou seulement 2 ou 3 dans le second, selon le moment

où l'instruction unique de Q est exécutée dans le temps d'exécution de P1 et P2.

De manière encore plus pernicieuse, une interactivité non coordonnée peut également se produire *au cours même* de l'exécution d'une instruction élémentaire de la procédure, comme dans cet exemple de Manna et Pnueli (1992, p. 102) :

```
when y == y do S.
```

Si l'environnement modifie la valeur de `y` au cours de l'évaluation de la condition `y == y` (celle-ci requérant en effet que le processeur inspecte la variable y à deux moments successifs) alors cette condition apparemment tautologique peut ne plus se trouver vérifiée, et l'instruction `S` ignorée, contre toute attente du programmeur.

L'interactivité non coordonnée présente d'autres problèmes encore. Dans le cas où l'environnement est lui-même constitué de procédures, cette interférence se produit si les temps d'exécution de ces unités computationnelles (par exemple des processeurs indépendants) peuvent se chevaucher (*overlap*). Par ailleurs, il peut se passer au contraire qu'une procédure avance beaucoup plus vite qu'une autre, ne leur permettant pas d'interagir correctement.

Pour ces raisons, le programmeur essaie en général d'éviter ces cas d'interactivité non coordonnée ou de les réguler. Ceci passe d'abord par l'organisation modulaire de la mémoire, de telle sorte que chaque processus travaille sur des données protégées. Concernant les données partagées, des mécanismes de protection temporaire se déclenchent quand un processus doit y accéder. Enfin, si une procédure doit réagir aux messages d'urgence évoqués plus haut, on programme alors une « méta-procédure », appelée souvent *ordonnanceur* (*scheduler* en anglais) qui en contrôle pas à pas l'exécution, ainsi qu'une autre sous-procédure, appelée *moniteur* qui écoute en permanence l'environnement. Si un signal est reçu par le moniteur, l'ordonnanceur décide s'il doit laisser la procédure courante s'exécuter, ou en lancer une autre. Enfin, concernant le chevauchement et les rythmes différents d'exécution des procédures, on tente de les réguler par un méta-ordonnanceur qui impose, sinon la synchronisation forte, du moins l'entrecroisement (*interleaving*) de l'exécution des différentes procédures présentes dans l'environnement : non seulement aucune instruction atomique ne peut en chevaucher une autre, mais il y a un certain équilibre dans le progrès des différentes procédures en jeu, qu'on désigne sous le terme général *équité* (*fairness*)[7].

Toutes ces raisons font que l'interactivité coordonnée est à la fois souhaitable,

7. Voir les développements du chapitre 2 de (Manna et Pnueli 1992).

et en général possible. La théorie formelle la plus connue, le π-calcul développé par Milner, se place donc d'emblée dans ce cas de figure : toute interactivité entre deux processus est modélisée par un canal de communication explicitement établi entre eux, impliquant la simultanéité de l'envoi et de la réception du message. Les ressources partagées, comme la « mémoire » ou les bases de données, sont elles-mêmes des processus dont l'accès est régulé par des canaux. Il est cependant important de bien souligner qu'il s'agit là d'une simplification d'un concept beaucoup plus général d'interactivité entre processus.

§ 221. Insuffisance de la représentation classique de l'interactivité

L'interactivité est l'objet de la théorie de la concurrence, un pan d'importance croissante de l'informatique théorique. La singularité des procédures interactives n'est en effet apparue que progressivement. Cela s'explique par le fait que du point de vue mathématique – celui de Turing et Church – les procédures fonctionnelles sont suffisantes pour représenter l'interactivité coordonnée. En d'autres termes, les procédures interactives ne peuvent pas calculer plus de fonctions que les procédures fermées décrites par le λ-calcul ou les machines de Turing.

Il est en effet possible de simuler toute procédure interactive par une procédure fonctionnelle classique. Il suffit pour ce faire de nous placer dans le cas d'une interactivité coordonnée, suite à notre discussion de la section précédente. Dans le formalisme des machines de Turing, cela peut se voir en concevant chaque donnée fournie par l'environnement comme un branchement dans l'exécution de la procédure. Comme il s'agit d'une interactivité coordonnée, cette donnée est requise par la procédure elle-même, à un moment bien déterminé de son exécution : il est donc possible de localiser la configuration de la machine où ce branchement devra se produire. Par ailleurs, le nombre de branches, c'est-à-dire de suites possibles de la procédure, doit être fini, si on fait l'hypothèse qu'on ne peut programmer un nombre infini de procédures. Ces conditions étant admises, on se trouve face à une machine de Turing dite non-déterministe, c'est-à-dire qui admet plusieurs transitions possibles pour chaque configuration. Or une telle machine peut être simulée par une machine de Turing classique[8].

D'un point de vue théorique donc, la classe des procédures interactives, celle des procédures non-déterministes et celle des procédures fonctionnelles classiques

8. Voir par exemple (Sipser 1997, p. 138-139) pour une démonstration de ce point.

ont un même domaine de calculabilité. L'interactivité peut toujours être ramenée à un cas fonctionnel, en envisageant tous les cas possibles d'interaction.

Malgré cette réductibilité de principe, de nombreux théoriciens ont progressivement affirmé que l'interactivité était un phénomène original, qui ne pouvait être adéquatement saisi par la théorie classique de la calculabilité. Ainsi Milner, dans son discours de réception du prix Turing, insiste sur le caractère progressif avec lequel cette conviction s'est affirmée :

> Au cours des années 70, je suis devenu convaincu qu'une théorie de la concurrence et de l'interaction nécessitait un nouveau cadre conceptuel, et pas seulement un raffinement de ce que nous trouvons naturel pour le calcul séquentiel. Souvent, les expériences qui provoquent une conviction ne sont ni planifiées ni profondes[9].

Le premier argument avancé pour justifier ce besoin est pragmatique. Au cours des années 1970-1990, les systèmes informatiques interactifs se sont développés à grande vitesse, par opposition à l'exécution séquentielle et fonctionnelle des premières décennies :

> Les théories surgissent généralement pour expliquer la pratique. Récemment, il y a eu un changement radical dans la pratique informatique ; en raison des progrès technologiques, les systèmes interactifs deviennent la norme plutôt que l'exception, et notre vision globale de l'informatique a changé en conséquence. La nouvelle technologie a créé le besoin d'étendre notre théorie des processus algorithmiques séquentiels aux systèmes où l'interaction joue un rôle significatif et même dominant[10].

La théorie classique de la calculabilité ne répond pas aux besoins de ces nouvelles pratiques car elle ne modélise pas leurs phénomènes ou objets d'intérêt. Nous pouvons expliciter cette assertion en trois points.

1. *Comportement* vs *résultat*. D'abord, comme le disent Manna et Pnueli, tandis que l'objet des procédures fonctionnelles (qu'ils appellent transformationnelles) est de *transformer* des données d'entrée en résultats de sorties, celles qui sont interactives (qu'ils appellent réactives) ont plutôt pour objet de *se comporter* d'une certaine façon, souvent d'une manière indéfinie :

> Un programme transformationnel est le type de programme le plus classique, dont le rôle est de produire un résultat final au terme d'un calcul. Par conséquent, la vision utile d'un programme transformationnel est de le considérer comme une fonction (éventuellement multivaluée) d'un état initial à un état final ou un résultat final. […] Le rôle d'un programme réactif, en revanche, n'est pas de produire un résultat final

9. (Milner 1993, p. 80).
10. (Milner 1999, p. x). Voir également (Wegner 1997, p. 80).

> mais de maintenir une interaction continue avec son environnement. Des exemples de programmes réactifs sont des systèmes d'exploitation et des programmes contrôlant des processus mécaniques ou chimiques, tels qu'un avion ou un réacteur nucléaire. Certains programmes réactifs ne doivent pas se terminer. Ils ne peuvent pas être spécifiés par une relation entre les états initial et final, mais doivent être spécifiés en termes de comportement continu[11].

Si on définit le comportement d'un processus comme le cœur de leur étude théorique, alors la différence entre processus fonctionnels et processus interactifs saute aux yeux, comme Wegner l'exprime avec force :

> Les algorithmes sont métaphoriquement idiots (*dumb*) et aveugles car ils ne peuvent pas s'adapter de manière interactive pendant qu'ils calculent. Ils sont autistes en ce qu'ils exécutent des tâches selon des règles plutôt que par l'interaction. En revanche, les systèmes interactifs *sont ancrés dans une réalité externe* à la fois plus exigeante et plus riche en comportement que le monde basé sur des règles des algorithmes non interactifs. […]
>
> Les difficultés croissantes que rencontrent les technologies logicielles sont dues au fait que la programmation de grands systèmes est intrinsèquement interactive et ne peut être exprimée ou réduite à la programmation étroite [de fonctions]. Le comportement des systèmes de réservation des compagnies aériennes et autres systèmes embarqués ne peut être exprimé par des algorithmes[12]. *(Nous soulignons)*

2. *Représentation locale* vs *globale*. Ce premier point se traduit par la lourdeur et l'inadéquation de la description classique de ces phénomènes. Comme le dit Milner, tout système interactif peut être modélisé par une composition séquentielle de fonctions qui transforment un état en un autre, ce qu'il exprime par l'identité sémantique suivante :

> Significations du Programme = Mémoires $\rightarrow$ Mémoires[13]

Le terme *mémoires* est mis ici au pluriel, car dans le cas de plusieurs processus interagissant ensemble, il faut considérer ensemble leurs états pour décrire ce qui se passe. Par exemple, si deux personnes possédant un compte bancaire joint effectuent chacune un transfert vers un compte personnel, on aura le comportement suivant du système, représenté par la valeur initiale du compte joint (ici $A = 300$ initialement), et celles des deux comptes individuels (ici $B1 = B2 = 0$ initiale-

11. (Manna et Pnueli 1992, p. 3).
12. (Wegner 1997, p. 82).
13. (Milner 1993, p. 80).

ment) :

$$\{A = 300; B1 = 0; B2 = 0\} \xrightarrow{\textit{transfert B1}} \{A = 200; B1 = 100; B2 = 0\}$$
$$\xrightarrow{\textit{transfert B2}} \{A = 100; B1 = 100; B2 = 100\}$$

On peut décrire l'état du système comme l'union des états de chacun des trois comptes, mais une telle description ne serait pas conforme aux pratiques de la programmation, qui insiste sur la modularité. Supposons qu'il y ait une dizaine de procédures bancaires (ouverture, clôture, transfert, crédit, etc.) pouvant agir sur un nombre fini de comptes bancaires ; décrire et vérifier un processus global requerrait de se représenter toute la combinatoire possible de processus concurrents et à fixer leurs incompatibilités et leurs priorités relatives. La modification d'une seule procédure pourrait requérir de vérifier tout le système à nouveau. Cela n'est pas gérable ; ce qu'il est nécessaire de faire au contraire, c'est concevoir des mécanismes de protection globaux concernant l'accès des procédures aux comptes bancaires, ce qui permet ensuite de vérifier chaque procédure individuellement.

La représentation des processus concurrents dans la théorie classique impose en quelque sorte de les considérer tous ensemble, comme une seule gerbe d'épis, qui serait ensuite débitée en tranches horizontales, au lieu de considérer chaque épi dans son individualité verticale et ses interactions fines avec ses voisins.

Mais il n'y a pas que cela. Une représentation globale de processus concurrents ne peut rendre compte que difficilement des non-déterminismes temporels qui peuvent survenir. Nous avons déjà mentionné le non-déterminisme lié à la variabilité des interactions, qui impose que pour chaque transition entre états, toutes les possibilités soient envisagées. Un système global doit lui-même ordonnancer les interactions afin de les traiter séquentiellement. Il va par exemple utiliser le mécanisme d'interactivité synchronisée le plus simple et « donner la main » à chaque processus successivement. Dans le cas précédent, si B1 et B2 sollicitent le transfert « en même temps », il va devoir choisir de faire passer l'un avant l'autre, ce qui peut modifier l'état global du système, si par exemple le compte joint n'est pas suffisamment approvisionné pour satisfaire les deux demandes. Or un système concurrent est non-déterministe à cet égard : il est important de pouvoir étudier tous les cas de figure. Afin de satisfaire cette exigence, il faut *a minima* que la simulation de la concurrence considère, pour une seule transition globale de n processus (où chaque processus avance d'une seule opération), $n!$ ordres possibles de cette transition. Et encore, cet éventail de possibilités ne rend pas compte du

non-déterminisme qu'on observe vraiment dans les systèmes concurrents, où par exemple, certains processus peuvent avancer plus vite que d'autres, avoir priorité, etc.

Par-dessus ces problèmes, un troisième non-déterminisme complique encore la représentation « classique » de la concurrence. Il est en effet souhaitable, dans les représentations de systèmes interactifs, de pouvoir introduire des *nouveaux* processus au cours de l'exécution. Ceux-ci sont générés par les processus existants mais ils ont une exécution indépendante d'eux, contrairement aux appels de sous-procédures de la théorie classique, qui interrompent l'exécution de la procédure courante jusqu'à leur terminaison. Par exemple, si le compte joint reçoit une demande qu'il ne peut satisfaire, il peut, au lieu d'émettre un refus, enclencher une demande d'autorisation de découvert. ll y a là création d'un nouveau processus auquel l'ordonnanceur central doit faire une place. De tels embranchements peuvent sans doute être représentés en théorie classique, cependant au prix d'une abstraction supplémentaire à effectuer sur l'espace des fonctions de transition, qui les rend encore plus difficiles à manipuler.

3. *Difficulté à traiter les problèmes spécifiques des systèmes interactifs*. Le plus important est que cette représentation complexe n'est pas adaptée à l'étude des propriétés spécifiques des systèmes concurrents. Il s'agit par exemple de s'assurer qu'un résultat recherché *finira toujours par arriver* (vivacité, *liveness*), ou qu'un événement indésirable *n'arrivera jamais* (sûreté, *safety*). Parmi ces événements indésirables, le blocage (*deadlock*) désigne la possibilité que tous les processus concurrents d'un système soient bloqués dans leur progression. Cela peut survenir, par exemple, si chacun attend d'un autre une information, et cela de manière à former une boucle. Un panorama de ces propriétés est donné par Manna et Pnueli (1992, chp. 4). On voit de suite que leur étude requiert, comme nous l'avons dit, de prêter avant tout attention à la succession et l'interdépendance des transitions, ce que la théorie classique de la calculabilité ne sait absolument pas faire.

§ 222. Kay, Hewitt, Milner

Ces phénomènes ont progressivement amené les théoriciens de l'informatique à élaborer de nouveaux formalismes pour les décrire. Un réseau dense d'influences réciproques se met en place au cours des années 1980 – on peut l'observer par exemple dans les « Remerciements » de Hewitt, Bishop et Steiger (1973, p. 244), la mention des « Sources et travaux apparentés » de Milner (1993, p. 83), et les

discussions bibliographiques situées au terme de chaque chapitre de Manna et Pnueli (1992). Ce bouillonnement théorique s'observe par la diversité des termes par lesquels sont désignés les objets d'étude : *processus, programmes réactifs, agents, objets, acteurs*, et par le chevauchement conceptuel de plusieurs notions étudiées : *concurrence, parallélisme, interactivité, réactivité*.

De manière intéressante pour notre propos, on peut dans ce réseau d'influences discerner une filiation – parmi toutes celles qui sont possibles – qui va de Smalltalk, un langage de programmation élaboré par Alan Kay en 1972, au cadre de conception et de spécification des Acteurs (*Actors*) de Carl Hewitt, qui se développe à partir de l'année suivante, jusqu'aux algèbres des processus qui apparaissent à partir de la fin des années 70, comme CSP, CSS ou le π-calcul. L'intérêt de cette influence spécifique est d'abord qu'elle est explicitement mentionnée : Alan Kay est la première citation d'importance dans les *Remerciements* de Hewitt, et Milner[14] reconnaît l'inspiration importante que fut pour lui le cadre des Acteurs. Cette influence va en quelque sorte de la pratique vers la théorie, puisqu'elle débute par un langage de programmation concret (Smalltalk), qui passe par un cadre de conception et de spécification (les Acteurs), pour aller vers un cadre mathématique pur (les algèbres de processus). Ce chemin illustre la découverte progressive et empirique de la théorie de la concurrence et des procédures interactives. Surtout, l'objet explicite de cette influence est la *radicalité* avec laquelle ces trois formalismes entreprennent de repenser la notion de calcul.

En effet, une manière naturelle d'étudier les procédures interactives serait *d'étendre* le formalisme classique du calcul. On a ainsi développé les notions de machines de Turing persistantes[15] ou enrichi les méthodes formelles de vérification des programmes par la logique temporelle[16]. Par contraste, le point commun des trois formalismes cités est de vouloir *refonder entièrement* la notion de calcul à partir de celle d'interactivité. Cela doit s'entendre littéralement : ils sont tous trois fondés sur un seul concept clé, comme le λ-calcul est entièrement fondé sur la λ-abstraction. Il s'agit des notions d'*objet* pour Smalltalk, d'*acteur* pour le formalisme de Hewitt, et de *réaction* pour le π-calcul. Il est utile de citer explicitement Milner à ce sujet :

> Le λ-calcul pur est construit avec seulement deux sortes de choses : des termes et des variables. Peut-on être aussi économe pour un calcul de processus ? Carl Hewitt,

14. (Milner 1993, p. 86).
15. (Goldin 2000 ; Goldin et al. 2004).
16. (Manna et Pnueli 1992).

> avec son modèle des Acteurs, répondit à ce défi il y a longtemps déjà. Il déclara qu'une valeur, un opérateur sur des valeurs, et un processus devaient tous être le même genre de chose : un acteur. Ce but m'impressionna[17].

Cette radicalité provient du fait que les interactions ne peuvent être conçues comme des relations ou des rapports qui surviendraient « par-dessus » des objets immuables, que seraient les fonctions de la théorie classique de la calculabilité – comme des réactions chimiques qui agiraient au niveau moléculaire, mais qui ne pourraient corrompre les atomes qui en sont les acteurs. D'abord, comme le remarquent Manna et Pnueli (1992, p. 4), on peut construire des fonctions « classiques » à partir de procédures interactives. Plus fondamentalement, nous avons vu que l'interaction faisait qu'un processus pouvait contrôler le déroulement interne d'un autre, en modifiant par exemple une variable partagée, qui l'oriente vers un autre cours d'action.

Ces faits amènent Kay, Hewitt et Milner à penser l'interactivité comme l'unité fondamentale du calcul, à partir de laquelle on peut dériver celles de la théorie classique, comme *fonction*, *état*, ou *mémoire*. Ainsi, pour Kay, la programmation en Smalltalk se fait directement en construisant des objets, qui sont définis par leur état (lui-même une suite d'objets) et leur réaction aux messages qu'ils peuvent recevoir, réaction qui consiste à modifier leur état et à envoyer des messages à d'autres objets. Pour Milner, un λ-terme est un cas particulier d'interaction, et un automate fini est décomposé en autant de processus qu'il a d'états. Les transitions entre états sont des réactions. La mémoire est elle-même un processus, voire même, chaque cellule de mémoire. Aussi, dans ces formalismes, un objet aussi simple qu'une machine de Turing (en théorie classique) apparaît là comme étonnement complexe, puisque chacun de ses états et chaque cellule de son ruban peuvent être vus comme un processus en interaction avec tous les autres !

12.2 Le couplage entre programmes et situation

Les procédures interactives sont bien des objets de savoir, mais elles n'ont pas une forme fonctionnelle, puisque le calcul de la fonction peut être modifié par l'environnement (et le modifier à son tour) au cours de son exécution. Elles peuvent être certainement simulées par des fonctions, comme une machine de Turing déterministe peut simuler une machine de Turing non-déterministe. Cependant cette

17. (Milner 1993, p. 86).

simulation est une opération de *représentation* théorique de ce qui se passe vraiment, en général seulement accessible *a posteriori*. La systématisation se révèle ici impuissante à expliquer, prédire et contrôler des processus qui sont pourtant l'exécution de procédures.

Cette impuissance provient du fait que cette ratiocination fait appel sans cesse à son environnement pour progresser. Quand cet environnement est la situation même que l'investigateur cherche à contrôler (et non, par exemple un contexte informatique constitué d'autres processus), la procédure interactive se trouve ainsi synchronisée ou « ancrée » avec l'état réel de la situation.

Cette section est consacrée à développer cette idée et ses conséquences. Nous explicitons d'abord (§ 223) le sens que nous donnons à cette expression ; notre idée principale est que par cet « ancrage » les règles programmées deviennent *directement effectives dans la situation*. Nous exposons ensuite (§ 224) deux schémas courants d'un tel ancrage, celui du *registre* et celui du *flux de tâche*, que nous illustrons par des exemples. Nous montrons que l'idée de *règles directement effectives dans une situation* est la source des avantages que nous avions trouvés à l'informatisation lors de notre première analyse[18].

Notre développement suivant (§ 225) étudie cette idée dans le cas où ces schémas concernent des objets matériels, qu'on appelle alors systèmes cyber-physiques[*]. Nous reformulons alors l'idée d'*ancrage* comme celle d'un *couplage* entre les règles régissant le comportement de la machine (son programme), et les règles concernant le comportement désiré de cet objet. Cela nous permet d'élargir la première source d'effectivité que nous avions trouvé à l'informatique au chapitre précédent[19] – celle de permettre la conception de machines industrielles capables de toute forme calculable de production – à toute sorte de machine en général, industrielle ou non, matérielle ou non, selon le concept générique que nous en avions donné[20].

Nous terminons cette section (§ 226) par l'étude du paradoxe qui résulte de ces développements, à savoir que les machines informatiques, qui rendent possibles ces couplages entre systèmes de règles et objets de la situation, sont en quelque sorte escamotées et deviennent invisibles à l'investigateur, qui s'intéresse directement au comportement de ses objets d'intérêt et non plus à elles. Cette invisibilisa-

18. Voir plus haut, section 5.4.
19. Voir plus haut, § 208.
20. Voir plus haut, § 204.

tion doit être entendue en un sens assez littéral, puisque l'ordinateur tel que nous le connaissons, devient, soit un micro-contrôleur caché au cœur des mécanismes de l'objet cyber-physique, soit une machine virtuelle* dans l'informatique en nuages*, dans les deux cas physiquement inaccessible à l'investigateur. Cela résulte directement du fait que la programmation devient à présent directement *programmation de la situation elle-même*.

§ 223. Persistance et ancrage dans la situation

En quoi les tentatives de refondation de Kay, Hewitt et Milner nous intéressent-elles ? Après tout, il est courant que les formalismes se ré-expriment les uns les autres, et nous sommes accoutumés à ne plus interpréter ontologiquement les projets axiomatiques.

Le point central nous semble être situé dans la différence *épistémologique* qu'il y a entre *concurrence* et *interactivité*, suivant une remarque de Wegner[21], pour qui la seconde est plus fondamentale que première. En effet, la concurrence simple, sans interactivité, ne pose aucune difficulté théorique : des procédures s'exécutant concurremment mais dans leur indifférence réciproque pourraient tout à fait être étudiées par la théorie classique de la calculabilité.

Par ailleurs, quand on parle de concurrence, on prend d'emblée un point de vue global et surplombant sur les processus en cours d'interaction. On se situe dans une position « théorique », ou encore « spectatrice » dans le sens évoqué par Dewey[22]. Au contraire, quand on parle d'une procédure interactive, on prend le point de vue de *cette* procédure, qui interagit avec un environnement dont elle ne connaît que l'information qu'il lui envoie. On se situe donc d'un point de vue local : l'interactivité décrit simplement l'interdépendance entre l'exécution d'une procédure et sa réceptivité à des messages provenant d'autres processus. C'est donc l'interactivité qu'on programme d'abord, et la vision de la concurrence est un point de vue synthétique et global qui correspond à une supervision observant « d'en haut » cette interactivité se dérouler. La différence entre concurrence et interactivité est la même qu'il y a entre observer deux personnes en train de dialoguer et prendre part au dialogue soi-même.

Manna et Pnueli[23] affirment qu'on peut modéliser l'interactivité par la concur-

21. (Wegner 1997, p. 84).
22. Voir plus haut, § 179.
23. (Manna et Pnueli 1992, p. 4).

rence, en représentant l'environnement comme un processus concurrent, mais cela ne peut se faire que si on peut le décrire par une procédure. Or cela n'est pas possible lorsque l'environnement se trouve être *le monde réel*. Reprenons l'exemple déjà évoqué du transfert d'argent entre comptes bancaires. On peut parler de situation concurrente si on suppose, pour simplifier, que ces comptes sont hébergés par le même établissement bancaire, et qu'ils sont chacun représentés, dans le système d'information de celui-ci, par un processus persistant qui, entre autres, conserve l'information du montant détenu. Lors d'un transfert, ces comptes interagissent entre eux, avec pour conséquence que leurs valeurs respectives en seront modifiées, pour l'un positivement, et pour l'autre négativement, mais ces valeurs continuent de persister simultanément dans le système d'information.

Lors d'un retrait de liquidités cependant, la situation est différente. Le processus associé au compte bancaire est décrémenté de 100€, mais il n'y a nul processus informatique qui garde trace de ce qui advient de cet argent dans la poche de son bénéficiaire. Il y a eu interaction, mais pas concurrence entre le processus informatique et son environnement.

Cet exemple nous permet d'illustrer une différence essentielle entre interactivité et concurrence. L'opération de retrait dépend de l'interrogation du processus informatique associé au compte bancaire, qui permet de vérifier que les fonds détenus sont suffisants, que le compte n'est pas bloqué, etc. Cette relation *ancre* ce processus dans l'expérience, pour reprendre l'expression de Wegner citée plus haut[24]. Le processus aboutit à une décision irrévocable du point de vue informatique, de donner 100€ en liquide au bénéficiaire. Cela veut dire, dans les termes de Dewey, qu'il est un jugement qui a des conséquences réelles, ou en tous les cas aussi réelles que sont les 100€ dans la poche du bénéficiaire. Une implication de cette réalité est que ce processus ne peut être quelconque, de sorte qu'il pourrait être exécuté depuis n'importe quelle machine. Il est au contraire critique pour l'établissement bancaire que ce processus soit *uniquement* identifié au sein de son système d'information, et qu'on puisse à tout moment *tracer* toutes les opérations s'y rapportant.

Si on n'observait jamais au contraire que des interactions entre processus concurrents dans un système d'information fermé, sans relation aucune avec des opérations réelles, il n'y aurait là aucun ancrage dans l'expérience. Il pourrait s'agir tout aussi bien d'une simulation ou d'un test d'une application bancaire. Toutes ces interactions, prises ensemble, pourraient être considérées par

24. Voir plus haut, p. 868.

l'observateur comme un seul processus fonctionnel, qui pourrait être exécuté depuis n'importe quelle machine.

Un processus informatique a donc autant de réalité que la situation avec laquelle il interagit. Il ne s'agit pas d'une assertion ontologique profonde concernant la réalité des processus informatiques, mais d'une considération logique, que l'observateur doit décrire ce processus selon les mêmes termes qu'il utilise pour décrire la situation. On ne parle ici que d'une réalité intentionnelle, celle de la situation dans laquelle une machine est intentionnellement insérée pour résoudre le problème visé par les acteurs humains. Ceux-ci peuvent certes avoir entre eux des malentendus ou des désaccords concernant ce que la machine est censée accomplir, mais cela montre bien que celle-ci se situe sur le même plan de réalité que la situation.

Ce principe vient compléter sur le plan matériel l'observation que nous avons déjà plusieurs fois faites sur le plan procédural, qu'un programme manipule directement les grandeurs pertinentes de la situation (ainsi le nombre d'entrées pour le contrôleur du portillon), et qu'un langage de programmation est une théorie des opérations *du domaine objet*, comme Catala qui parle le langage des juristes. Ainsi, le processus informatique qui enregistre les opérations du compte bancaire est bien le livre comptable qui fait foi vis-à-vis du système bancaire tout entier – et cela non pas comme une relation symbolique générale, mais comme *ce* processus-ci, instancié sur telle machine. On doit lui accorder autant de « réalité » qu'on en accorde au système bancaire, car s'il y a contestation sur ces opérations, c'est vers ce processus que se tourneront les garants du système pour inspecter les revendications de chaque partie.

De la même manière, le processus informatique qui joue contre nous aux échecs est bien notre adversaire, c'est-à-dire *ce* processus particulier. Il ne serait pas acceptable qu'il puisse être reprogrammé en cours de route. Dans le cas contraire, on pourrait suspecter d'avoir affaire à un « turc mécanique », l'automate factice du baron von Kempelen qui ravit les cours européennes du XVIII^e^ siècle, et qui était animé de manière secrète par un vrai champion d'échecs. Il était très important, lors des rencontres de 2016 entre AlphaGo, le programme d'intelligence artificielle élaboré par la société DeepMind pour jouer au Go, et Lee Seedol, le champion du monde à cette époque, que le programme soit lancé sur une machine scellée, afin qu'il ne puisse y avoir aucun doute sur le remplacement subreptice du processus par un autre au cours de la partie. C'est que, *dans la réalité intentionnelle* de ces

jeux et de ces championnats, il importe qu'on joue contre un même individu tout au long de la partie et du tournoi – qu'il soit une personne ou un agent robotisé n'y change rien.

Il est important de bien marquer ici ce qui caractérise un processus interactif ; qu'il ait un ancrage et un poids dans la situation ne veut pas dire qu'il y accomplit des choses importantes. Un processus purement fonctionnel peut avoir une influence forte sur un environnement donné, par exemple une fonction qui calcule le plan de production d'une usine, ou même une machine réelle qui transforme des matières premières en un produit fini. Il n'y a cependant pas d'ancrage, car justement ces calculs et cette production auraient pu être réalisés par n'importe quelle autre machine similaire, à n'importe quel autre moment. Comme nous l'avons dit, l'investigateur qui interroge un programme fonctionnel fait office d'*interrupteur herméneutique* entre la machine et la situation, qui se trouvent ainsi découplés intentionnellement[25].

La différence essentielle entre un processus fonctionnel et un processus interactif concerne donc leur temporalité. Le premier résout un problème sans interaction avec l'environnement externe à la machine, et n'intervient dans l'expérience qu'au moment où il produit un résultat, qui peut avoir des conséquences réelles à ce moment là. La durée de son exécution n'est pas pertinente. Si elle l'est, il faut lui adjoindre un chronomètre, qui est déjà une première forme d'interactivité. Au contraire, un processus informatique interactif est (dans l'intention même de sa conception) en interdépendance avec son environnement et *couplé* avec lui. Il acquiert par là la même réalité que lui durant tout le temps de son exécution.

Les jeux informatiques permettent de mieux appréhender les nuances de ce couplage de réalité entre les processus interactifs et leur environnement. Ils ont une position intermédiaire entre les processus fonctionnels fermés et ceux qui interagissent avec des environnements ouverts. Nous désignons par *jeux* une classe plus large que celle stricte des « jeux vidéo ». Elle comprend également, par exemple, les simulateurs de vol, qui ne sont pas développés à des fins ludiques mais d'apprentissage. Ces applications sont en général dotées d'une interactivité forte, souvent avec plusieurs participants humains. Les jeux dits massivement multi-joueurs rassemblent des milliers de personnes, et certains ont une persistance continue sur plusieurs années. La caractéristique de ces applications est cependant que les personnes humaines y endossent un rôle de joueur qui établit normalement une

25. Voir plus haut, § 59.

séparation stricte avec leurs autres activités. Ce qui se passe dans le jeu n'a pas normalement de répercussion ailleurs, ce qui permet de parler ici d'un environnement semi-ouvert. Le processus informatique interagit bien avec des personnes humaines, mais il a seulement autant de réalité que celle qu'elles attribuent au jeu, celle d'un espace fermé sur lui-même, d'évasion hors des règles de la vie réelle. Bien entendu, lorsque des transactions financières commencent à circuler au sein du jeu, ou lorsque celui-ci commence à influencer le comportement général de ses participants, alors on doit également lui accorder, dans cette mesure, une réalité socio-économique – tout comme on le fait, par exemple, pour d'autres institutions culturelles comme le sport.

Ainsi, lorsqu'ils ont une persistance longue (jeux multi-joueurs persistants, système d'information d'une organisation, services Internet comme Google,...), ces processus informatiques doivent être considérés comme des objets institués culturellement au même titre que les systèmes bancaires, les jeux, les organisations et les régimes politiques ou judiciaires. Ils sont des systèmes objectifs.

Cependant ils s'en distinguent nettement par deux propriétés : d'une part, dans les jeux informatiques les règles sont directement effectives – la machine est un arbitre indiscutable. D'autre part, ces règles peuvent être dynamiques, c'est-à-dire être configurées, changer lorsqu'on passe à un niveau de jeu plus difficile, s'adapter à la situation du joueur, etc. Ce sont ces deux propriétés qui, lorsqu'on *couple* des processus interactifs à des objets techniques ou institutionnels, modifient en profondeur ces derniers et, partant les pratiques humaines auxquelles ils participent.

§ 224. Le registre et le flux de tâches

Une des schémas les plus simples de programmation interactive est le *registre*, un processus informatique qui contient un certain nombre de règles (ou de conditions pour l'exécution de règles) à propos d'une certaine réalité, comme un compte bancaire, ou un salarié d'une entreprise. Un registre est simplement « réactif » en ce qu'il n'intervient dans une situation que lorsqu'il est sollicité – par exemple une demande de retrait concernant le compte bancaire. Il reste néanmoins « interactif » selon notre définition car ses propriétés importantes sont sa persistance dans le temps et son unicité relativement à l'objet visé. Il est important en général que l'intégrité du registre soit maintenue, c'est-à-dire que les conditions qu'il décrit ne puissent être falsifiées de manière indue. Il est donc lui-même associé à des droits

d'accès qui stipulent quels agents (informatiques ou humains) peuvent accéder à son contenu et quels agents peuvent le modifier.

Une précision terminologique est ici nécessaire : quand nous parlons de *processus*, nous n'entendons pas ce terme dans le sens technique qu'il peut avoir pour un système d'exploitation*, dont l'un des rôles principaux et d'organiser et maintenir de manière concurrente de telles unités d'exécution ; un processus, au niveau conceptuel auquel nous l'entendons, peut aussi bien être implémenté par un groupe de programmes que par des interactions entre programmes, ou par un fichier, ou tout simplement par un item dans une base de données. Il est avant tout un *système de règles dynamiques*, qui doivent devenir effectives dans certaines circonstances, grâce à une machine ancrée dans la réalité de la situation. Ainsi, lorsqu'une institution bancaire reçoit une demande de retrait pour l'un des comptes qu'elle héberge, les règles concernant celui-ci sont mobilisées grâce à un ordinateur précis, identifié par cette institution comme partie de son infrastructure informatique. La manière dont ces règles sont implémentées sur cet ordinateur a ici peu d'importance.

Par ailleurs, nous disons qu'un registre contient des règles ou des conditions pour l'exécution de règles ; cette distinction n'est pas non plus importante pour notre propos. Par exemple on peut décider de représenter un compte bancaire par un ensemble de données « statiques », comme le solde actuel du compte, en laissant d'autres processus interroger cette donnée et décider quoi en faire. Ou bien on peut représenter un compte comme un objet « actif » au sein du système bancaire, qui seul décide s'il doit autoriser ou non telle opération à son propos, en fonction de l'information fournie. Ces distinctions ne changent pas la nature de ce que nous appelons ici *registre*.

Le registre est une forme élémentaire et prégnante de processus informatiques, notamment dans les domaines institutionnels, puisqu'il enregistre des droits et des interdictions ; en particuliers, les bases de données dites *transactionnelles*, qui gardent trace de transactions *en cours* (commerciales ou non : il peut s'agir d'une demande de passeport), sont de telles collections de registres. Une fois cependant que la transaction est achevée, le registre devient une simple *archive*, dont l'unicité est moins importante. Nous avons rencontré déjà par deux fois de tels registres dans nos exemples : le contrôleur de portillon d'accès[26] est un registre, ainsi que la caisse enregistreuse (*cash register* en anglais)[27].

26. Voir plus haut, § 55.
27. Voir plus haut, § 88.

Lors de l'analyse de ce dernier exemple, nous avions identifié trois formes d'« avantages » procurés par l'informatisation. La source commune de ces avantages devient ici apparente : un registre ne fait que rendre la règle directement *effective*, au sens même où le Larousse donnait pour exemple de ce mot *« L'armistice sera effectif à 11 heures »*[28]. Pour mieux le voir, on peut prendre un exemple plus simple encore de registre, son modèle quasi-pur, celui du contrôle d'accès de personnes à un bâtiment protégé d'une institution.

Dans une version élémentaire, ce registre peut être une liste de personnes qu'un agent de sécurité peut consulter sur un support numérique. Ici, le registre a déjà la valeur de *contrôle*, puisque non seulement il contraint l'accès de personnes au bâtiment (par l'entremise de l'agent de sécurité), mais sous forme numérique, il peut lui-même être protégé de modifications indues (comme quelqu'un qui ajouterait un nom de manière subreptice sur une feuille de papier).

Une version plus sophistiquée du registre est de fournir un badge d'accès aux personnes autorisées, ce badge pouvant être activé ou désactivé aisément. Il y a ici *automatisation* des procédures de contrôle et de modification de droits, pour un gain de *productivité*. On peut y voir également un gain d'*exactitude*, puisque de nombreux risques de défaillance du contrôle sont résolus, par exemple une personne ayant oublié sa pièce d'identité, ou un gardien interprétant de manière trop laxiste les règles.

Une version plus avancée encore est de définir des rôles au sein de l'institution, chacun d'entre eux définissant un profil de droits – non seulement leurs accès aux bâtiments, mais également à des horaires particuliers de leur ouverture, à des salles particulières, à réserver des salles pour des réunions, à accéder à tels services informatiques et à telles informations, etc. Ici le badge (ou tout autre identifiant matériel, qui peut être biométrique) associe à chaque personne un ensemble des règles qui *guideront* sa conduite au sein de l'institution. Par là est illustré le second bénéfice de l'automatisation, celui de permettre de *nouvelles opérations* inaccessibles à une simple gestion manuelle. *Guider* doit être entendu dans le sens concret qui le rapproche de *rails*, ou de *rampe* : des dispositifs qui facilitent le progrès dans une certaine direction, mais l'y contraignent également.

Enfin, rien n'empêche que chaque sollicitation d'un droit soit enregistré, si bien que l'on dispose des *traces* des activités de l'individu au sein du bâtiment : à quelle heure il est arrivé en moyenne ce mois-ci, quelles applications informatiques il

28. Voir plus haut, § 3.

a consulté, etc. Cela ouvre à nouveau d'autre possibilités de droits dynamiques, comme un compte de cantine qui lui permet d'y enregistrer des tickets repas, etc.

On le voit : tous ces avantages tiennent à la seule idée d'une règle rendue directement effective dans une situation donnée, et qui lui en donne le *contrôle*. L'automatisation est simplement alors, non pas tant le remplacement d'opérateurs humains par des robots, que la *suppression* pure et simple de tâches adventices à la règle, chargées d'assurer son application et de gérer son évolution ; avec le badge d'accès, c'est en quelque sorte le bâtiment lui-même qui se protège de visiteurs non désirables. Enfin la *traçabilité* est rendue possible par le fait que la règle devient un objet à part entière dans la situation, et dont les interactions avec elle peuvent être observées et enregistrées.

Il est donc naturel que le registre puisse devenir lui-même *moteur* d'activités dans la situation. Il devient alors *flux de tâches* (*workflow*), qu'on peut voir comme la version interactive d'une procédure. Le flux de tâches se compose d'une séquence de modifications d'objets de travail, ainsi par exemple un document que plusieurs collaborateurs s'envoient successivement après y avoir chacun effectué leurs modifications. Ce problème est central dans les organisations, où par exemple une commande doit passer à travers divers départements (par exemple, commerce, service client, production, logistique, comptabilité) afin d'être honorée. Un flux de tâches est composé d'abord des données nécessaires à son traitement, par exemple pour une commande, le nom et l'adresse du client, la désignation du produit à fabriquer, etc. Il y a ensuite un registre, appelé *statut de commande*, qui doit à tout instant refléter l'étape actuelle de traitement, et se couple donc avec le processus concret que suit l'entreprise pour honorer la commande.

Le flux de tâches ne se réduit cependant à ce simple enregistrement du progrès d'une tâche collective. Il a une dimension proactive qui distribue le travail au fur et à mesure de son avancement. Par exemple l'expédition d'une commande génère une tâche de facturation. Il doit également gérer les exceptions, comme un retard de production. En d'autres termes, un flux de tâches dispose surtout d'une logique interne qui permet de représenter des arbres de décisions, des boucles itératives, etc. Il s'agit donc d'une procédure dans le sens le plus courant qu'on peut donner à ce terme. Cependant, comme le registre, il n'est pas nécessairement représenté par un processus dans le système d'exploitation, ni par le codage d'une procédure dans le langage de programmation, mais par un ensemble de programmes, ou par des interactions entre programmes, ou encore par des items dans des bases de donnée,

etc. Pour cette raison, nous utilisons également à son propos les termes de *système de règles dynamiques* et de *processus* plutôt que ceux de *programme* ou *procédure* interactifs.

§ 225. Les systèmes cyber-physiques

Le registre et le flux de tâches sont des formes simples de processus interactifs persistants, couplés à des objets *situés* dans l'expérience, quels que soient les systèmes de significations par lesquels on décrit ces objets. Jusqu'ici nous avons évoqué le cas d'objets institués – compte bancaire, contrôle d'accès, commande de production, facture. Lorsque ces objets sont matériels, de tout autres conséquences deviennent possibles, et on parle alors de *systèmes cyber-physiques* (*cyber-physical systems*).

Une forme importante de système cyber-physique est le *système embarqué* (*embedded system*). Il s'agit d'un micro-ordinateur à programme fixe (qu'on ne peut reprogrammer), qui est intégré à un objet physique dont il va contrôler partiellement ou totalement le comportement par une procédure interactive (c'est-à-dire un flux de tâche). L'objet commandé peut être une machine, comme un ascenseur, un missile, un réfrigérateur, une machine industrielle, mais il peut s'agir aussi de capteurs* et d'actuateurs* attachés à des objets ou des structures passives, comme des ouvrages d'art. Par exemple, le système embarqué surveillant une section d'un réseau d'eau actionne une valve lorsqu'elle enregistre une pression trop forte.

Cette idée a plusieurs dénominations, selon les perspectives dans lesquelles on se place. Outre l'expression de *systèmes embarqués*, on parle donc également de *systèmes cyber-physiques* lorsqu'on veut mettre en avant le *contrôle* qu'il opère sur l'objet physique, en repartant de l'étymologie de *cyber-*, qui signifie l'acte de gouverner. La catégorie des systèmes cyber-physiques est plus large que celle des systèmes embarqués, puisqu'y entrent aussi les objets commandés à distance ou qui font partie d'un réseau d'objets commandés de manière synchronisée. Des dénominations populaires sont celle d'*objets intelligents* (*smart things*) ou, lorsque ces systèmes sont dotés d'un module de communication à distance, d'*objets connectés* et d'*Internet des objets*.

Le terme *système* dans les deux premières expressions citées est révélateur. Même si dans le premier cas, c'est une machine physique qui est évoquée par le qualificatif *embarqué*, au fond ce qui est visé, c'est le *système de règles* qu'elle impose au comportement de l'objet cible. L'expression *système cyber-physique* est

à ce titre plus heureuse, car elle n'introduit pas de distinction artificielle entre une partie de l'objet physique qui serait « passive » et l'autre « active », et le conçoit d'emblée comme un tout régi par des règles, évoquées par *cyber-*.

Ces systèmes cyber-physiques ne sont d'ores et déjà plus une exception, ni même minoritaires. Ils sont la norme dans les environnements industriels et dans les transports (avions, trains, voitures). Ils sont bien sûr l'essence des drones, dont le développement rapide se constate dans tous les domaines de l'activité humaine. Ils se démocratisent dans les infrastructures, mais également dans les biens de consommation durables, comme les appareils électroménagers. Ils sont également présents dans la nouvelle catégorie des vêtements et accessoires technologiques, comme les bracelets numériques qui mesurent des signes vitaux (fréquence cardiaque, taux d'oxygénation, etc.) du porteur et les lui restituent sous forme synthétique.

L'idée de système cyber-physique n'est que l'aboutissement de celle de machine, telle que nous l'avons exposée au chapitre précédent[29]. Une machine, avions-nous dit, est tout phénomène de l'expérience dont l'interaction avec son environnement semble réglée ; un système cyber-physique est exactement cela, sauf qu'ici cette interaction est régie par des règles effectives, dans le sens que nous venons de dégager. Pour mieux le voir, il est utile de partir encore une fois des machines de transmission.

Nous avions qualifié ces machines de *simples* du fait de la pauvreté de leur environnement, qui consiste seulement en « barrières » qui laissent passer ou non l'eau dans les canaux, pour reprendre les termes de notre exemple hydraulique. Cet environnement est à la main d'un opérateur humain, qui peut par ce biais contrôler la transmission. L'environnement de systèmes cyber-physiques simples est également très pauvre, seulement leur interaction n'est pas en général dirigée vers un opérateur humain, mais vers une dimension mesurable de l'environnement par le biais d'un capteur. Ainsi, une valve hydraulique sera actionnée par un capteur* de pression, ou l'ouverture d'une barrière par un signal infra-rouge. Les capteurs peuvent devenir plus sophistiqués : par exemple un détecteur de vitesse dans un système de freinage pourra mesurer la vitesse de rotation d'une roue, mais également son orientation et surtout son blocage éventuel. Un système d'avionique envoie au pilote automatique les informations d'un grand nombre de capteurs, comme des gyroscopes et des accéléromètres, des GPS, des capteurs de pression

29. Voir plus haut, § 204.

d'air, de positions de pièces mécaniques comme les ailerons, d'état des réacteurs, etc. On voit ainsi qu'un système cyber-physique est une machine dont la sensibilité à l'environnement est filtrée par des capteurs qui construisent un espace de conditions disponible pour la formulation de règles inductives.

Dans tous ces exemples cependant, le filtrage est strictement défini relativement à des grandeurs physiques bien connues. L'environnement d'un avion, aussi complexe soit-il, est cependant maîtrisable. Mais lorsqu'il s'agit par exemple de reconnaissance d'images, et plus généralement de schémas (*patterns*) dans un environnement « bruité », le système cyber-physique acquiert une autre dimension encore, où des données « brutes » doivent être comparées à des « modèles ». L'idée de voiture autonome illustre bien la complexité d'interpréter des millions de signaux visuels et sonores (des piétons hésitant à traverser, un coup de sifflet, un sac plastique rouge accroché à un poteau, une voiture avec son clignotant mis à mauvais escient, un marquage au sol effacé, etc.) pour prendre des décisions en temps réel. Ce qu'on tente de faire, c'est remplacer l'investigation active que réalise tout conducteur au volant – aller à sa destination, en résolvant au passage des dizaines de micro-situations problématiques comme celles que nous venons d'évoquer – par une procédure unique qui récapitulerait toutes les investigations possibles de cette classe. La difficulté essentielle ici n'est pas de déterminer les règles de conduite, mais d'organiser toutes les conditions possibles de la conduite en un espace qui les hiérarchise et qui permette de définir de manière inductive des règles à leur égard. Il s'agit sans doute là d'une des entreprises de systématisation *du monde réel* parmi les plus ambitieuses qui soit en cours aujourd'hui. Ainsi par exemple, une des propositions d'un projet actuel de véhicule autonome est de « lire » la situation relativement à un modèle organisé en six couches, ainsi qu'illustré dans la figure 12.1, concernant la géométrie et les marquages de la route, les structures de bord de route, leurs modifications temporaires, le trafic et plus généralement les objets dynamiques, la météo et enfin l'information reçue numériquement.

Ainsi on peut décrire un système cyber-physique comme un phénomène de l'expérience dont les interactions intéressantes avec l'environnement sont asservies à un contrôle de transmission. Les signaux de cette transmission ne sont plus déterminés par un opérateur humain ou par une autre transmission, mais par l'environnement physique lui-même, par le biais d'une interface qui peut être enrichie indéfiniment. Comme nous l'avons vu, elle n'est nullement limitée à la captation de données « sensorielles » ou « physiques ».

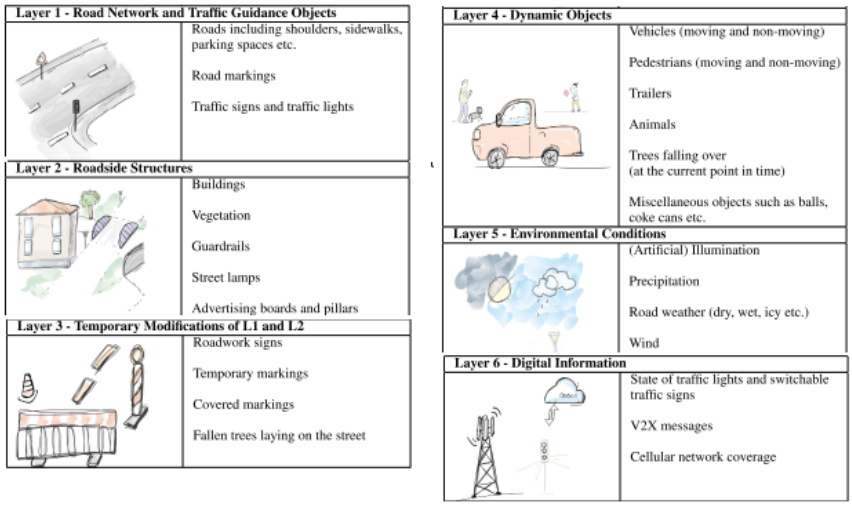


FIGURE 12.1 – Le modèle 6LM d'interprétation d'une situation de conduite. Source : (SCHOLTES et al. 2021)

On peut également définir un système cyber-physique d'une autre manière, qui est plus pertinente encore pour notre propos, en disant qu'il représente un *couplage entre un système de règles et un objet physique*. Dans cette définition, c'est l'intention primaire du concepteur du système qui est mise avant, contrôler selon des règles le comportement de l'objet cible ; le contrôle de transmission est omis de la définition, car il est simplement accessoire à cette intention.

On peut utilement comparer la notion de *couplage* que nous utilisons ici à celle de *jumeau numérique* (*digital twin*) qui désigne la réplique par un programme informatique de l'apparence et de certains comportements d'un objet ou d'une personne, de telle manière que la réplique et sa cible soient constamment *synchronisés* : le programme se met à jour régulièrement en fonction des données qu'il reçoit de l'objet par des capteurs*, et dans certains cas, il peut diriger son comportement ou lui envoyer des messages. Ce concept est issu de l'informatique industrielle, où il s'agissait initialement d'automatiser la planification fine de la maintenance grâce à l'analyse immédiate de données captées en temps réel sur les machines[30].

Par contraste, ce que nous appelons *couplage* ne désigne pas d'abord ni essentiellement l'appairage synchronisé d'un programme et d'un objet physique. Il ne se fait pas avec de l'information, ni avec du logiciel, ni avec des puces électroniques, mais tout simplement avec des règles opératoires que nous savons, d'une manière

30. (D. JONES et al. 2020).

ou d'une autre, inscrire dans les lois fondamentales de son comportement, en général au niveau des mécanismes internes qui transmettent le mouvement ou l'énergie entre ses parties. Il n'y a pas de monde numérique dans lequel habiteraient ces jumeaux numériques. Il n'y a qu'un seul monde, et il n'y a qu'une seule manière de l'appréhender, l'expérience. Mais notre expérience n'acquiert de sens que par des systèmes de significations qui dénotent les règles par lesquelles nous savons décrire le comportement régulier des phénomènes. La programmation a seulement trait à notre capacité d'intervention sur ces règles.

Avec la notion de *système cyber-physique*, la machine programmable nous apparaît dans toute sa généralité. Au chapitre précédent, nous avions identifié comme première source d'effectivité des ordinateurs la capacité qu'ils avaient à prendre le rôle de contrôleurs universels des machines industrielles[31], magnifiant par là les différents bénéfices d'automatisation (productivité, flexibilité, exactitude), de contrôle et de traçabilité de celles-ci[32]. On voit à présent que nous pouvons étendre le domaine de cette effectivité à tout type d'objet que nous parvenons à coupler avec un système de règles, et que les différents bénéfices rappelés à l'instant sont simplement la conséquence de ces règles devenant directement effectives dans l'expérience.

Nous pouvons l'étendre encore davantage, si on accepte, ainsi que nous l'avons proposé[33], qu'une machine ne désigne pas seulement des phénomènes du monde « physique », mais également tous les objets du monde humain dont nous connaissons les lois d'interaction avec leur environnement institutionnel – ainsi un compte bancaire, un portillon d'accès, la caisse d'un magasin, une plateforme logistique ; toutes ces « machines », dont l'effectivité dépendait jusqu'ici de personnes se comportant selon leurs règles, acquièrent par leur couplage à des processus interactifs une effectivité directe qui leur permet de réaliser les *mêmes* bénéfices que les machines industrielles – comme nous l'avons vu plus haut avec l'exemple du badge d'accès. On pourrait alors parler, par extension de l'expression de *système cyber-physique* aujourd'hui commune pour les objets matériels couplés à des systèmes de règles dynamiques, de *système cybernétique* en général pour tous les objets ainsi couplés, qu'ils soient matériels ou institués. Un tel choix terminologique cependant relierait notre réflexion à l'intuition cybernétique originelle de Norbert Wiener, ce

31. Voir plus haut, § 208.
32. Voir plus haut, section 5.4.
33. Voir plus haut, § 204.

qui exigerait une étude complémentaire pour éviter des contresens. Nous préférons opter pour l'expression plus neutre d'*objet interactif*, qui désigne donc, entre autres, les systèmes cyber-physiques.

§ 226. L'invisibilisation des machines

Dans la programmation et l'usage d'applications fonctionnelles, l'ordinateur se situe face à l'investigateur humain et lui est bien visible : c'est *lui* que ce dernier programme et c'est *lui* qu'il interroge afin qu'elle lui fournisse des réponses à ses questions, réponses qu'il aura alors la charge d'interpréter pour prendre une décision. L'investigateur humain, avons-nous dit, a alors le rôle d'un « interrupteur herméneutique » entre la machine et la situation. Lui seul décide de quelle manière elles interagiront.

Il en est tout autrement avec la programmation interactive, où la machine accompagne des objets de l'expérience technique et institutionnelle dans la réalisation de leurs propres règles internes. Elle s'efface ainsi derrière eux et devient en quelque sorte invisible. Ce n'est pas la machine qu'on programme, mais le système de sécurité, le fonctionnement de la mine, les procédures de vote électronique. Il n'y a plus là nulle métaphore, car c'est bien cela qu'on visait de faire depuis le début avec ces systèmes techniques et institutionnels, *i.e.* déterminer les systèmes de règles auxquelles ils devaient obéir.

Il convient de comprendre ce phénomène d'effacement assez littéralement, comme l'*invisibilisation* des machines. Cela est patent dans les systèmes cyber-physiques, où elles se font microscopiques pour s'insérer au cœur du mécanisme de l'objet et faire corps avec lui. Il n'y a pas lieu de distinguer entre le portillon d'accès et son contrôleur, car le second ne fait qu'exécuter la logique interne du premier[34]. De la même manière, dans les réseaux d'eau, les contrôleurs numériques peuvent être vus comme une simple évolution technique des valves hydrauliques, car ils permettent de mieux programmer leur comportement optimal[35]. Ainsi s'attend-on de plus en plus à ce que les objets techniques réagissent de manière toujours plus fine aux circonstances selon les règles *que l'on serait en droit d'attendre d'eux*, c'est-à-dire qu'ils soient *intelligents* dans le sens même où Aristote parlait des trépieds d'Héphaïstos, « capables sur une simple injonction, ou même pressentant ce qu'on va leur

34. Voir plus haut, § 54.
35. (Acuña-Bravo et al. 2017).

demander, d'accomplir le travail qui leur est propre[36] » – et on s'étonne lorsqu'ils faillissent – comme lorsqu'on perd ses clés et qu'il n'y a aucun moyen de les « appeler », ou qu'un four ne s'arrête pas de lui-même s'il reste allumé trop longtemps. Ce à quoi nous nous habituons rapidement, c'est à attendre du monde technique environnant qu'il se comporte selon les règles adéquates, à quelque profondeur que ce soit. Ces règles ne sont pas celles d'un ordinateur externe à l'objet, mais *celles de l'objet lui-même*, tel que nous le concevons.

C'est cependant dans les objets institués (un compte bancaire, un droit d'accès, etc.) que ce phénomène d'invisibilisation est le plus patent, puisqu'il conduit à rendre manifeste leur caractère procédural pur – ils ne sont plus rien d'autre que des procédures en cours d'exécution sur des ordinateurs distants. Dans le cas de l'informatique en nuages*, ces services sont « quelque part », exécutés par des machines virtuelles elles-mêmes instanciées dynamiquement dans des réseaux de serveurs physiques, qui sont seulement localisables par leurs administrateurs grâce à des systèmes d'adressage, des sortes de carnets d'adresses dynamiques. Chacune de ces couches d'infrastructure opère elle-même dans l'ignorance du fonctionnement de la couche inférieure, selon le principe de division du travail informatique. Il y a là une invisibilisation de l'infrastructure, qui l'escamote aux yeux de l'utilisateur-programmeur et lui fait croire à des « espaces virtuels » qui existeraient par eux-mêmes, comme si par exemple les téraoctets de messages, de photos et de vidéos répliquées et retransmis indéfiniment par les réseaux sociaux n'avaient pas d'ancrage matériel et une empreinte dans le monde – impression renforcée par l'apparence de gratuité de la plupart de ces services.

Cette illusion d'immatérialité doit certainement être combattue, ne serait-ce que pour des raisons environnementales, et plus généralement encore pour des raisons de contrôle démocratique sur une infrastructure critique pour le fonctionnement de nos sociétés. Cependant cette invisibilisation n'est pas la simple résultante de raisonnements économiques et techniques qui conduisent aux choix d'architecture que nous avons exposés. Elle participe à l'idée régulatrice de ce qu'est un système objectif, c'est-à-dire un ensemble de procédures actives dans le monde *via* des objets phénoménaux, que l'on peut solliciter pour réaliser des objectifs.

Il y a cinquante ans, une agence bancaire était dotée d'un guichet « Crédits » où un employé recevait les demandes de prêts et en faisait un premier examen avec le client, avant de les envoyer à un service central chargé de réaliser des investi-

36. *Pol.* I.4, 1253b33.

gations complémentaires et éventuellement de l'accepter, suite à quoi, le prêt était mis en œuvre dans les comptes de la banque. Le flux de tâches (la procédure) de prêt était déjà l'essentiel de ce qu'on pouvait observer empiriquement à travers les diverses activités des différents employés. L'invisibilisation de ce flux, aujourd'hui largement automatisé pour une grande part des prêts bancaires, n'est pas sa dénaturation, mais au contraire participe du même mouvement d'essentialisation des processus qu'observait Marx à propos de l'industrialisation de la production[37]. Simplement, ce flux de tâches bancaire n'a pas même besoin d'une machine pour réaliser son produit (le prêt), contrairement à une machine de production, qui reste visible car elle a une fonction matérielle. L'ordinateur ne sert qu'à héberger le flux de tâches : il est donc normal qu'il soit escamoté dans les coulisses de l'informatique en nuage.

12.3 La programmation de systèmes interactifs

Lorsque la programmation devient programmation directe d'objets de la situation, qu'il s'agisse d'objets techniques ou institués, il y a une logique interne à ce que ce soit *la situation tout entière* qui devienne programmée. La programmation d'objets devient programmation de systèmes, qui édictent des règles effectives auxquelles doivent se conformer toutes les procédures particulières, mais sur lesquelles elles peuvent également s'appuyer. Il ne s'agit plus dans ce type de programmation de résoudre des problèmes particuliers, mais de préparer des environnements systématiques pour la résolution de problème (§ 227). Nous exposons ensuite deux schémas d'interactivité par lesquels cette programmation de systèmes peut être exprimée. Il y a d'abord celui d'*espace de travail* qui, assez classiquement, fournit un cadre commun, normatif et rigide, à des classes données de registres et de flux de tâches – ainsi l'ERP* (§ 228). Par contraste, dans le *réseau de services* l'essentiel de l'interaction se produit entre flux de tâches automatisés, plutôt qu'avec des interlocuteurs humains ; le réseau se contente d'assurer la bonne interconnexion de ces services mais ne les administre pas (§ 229).

Dans la seconde partie de cette section, nous mettons en évidence la dynamique auto-reproductrice de cette systématisation. En effet, la contrepartie de l'adaptation fine des règles à la diversité des situations est que leur exécution par des interlocuteurs humains devient difficile voire dangereuse : l'interconnexion et l'intégra-

37. Voir plus loin, § 231.

tion des services entre eux devient de ce fait une nécessité, rejetant les personnes à la périphérie de la situation, leur assignant le rôle de purs consommateurs des services rendus par le système (§ 230). Il y a dans cette *désintermédiation* une deuxième forme d'opacité épistémique engendrée par la programmation interactive (§ 231).

§ 227. La résolution partielle de problèmes

Dans le flux de tâches, l'ordonnancement est *local* : il peut « décider » quels messages reçus ou tâches internes traiter en priorité, mais il n'a pas lui-même le pouvoir sur sa propre priorité au sein de la situation. Ce principe de localité est affirmé avec force par Hewitt[38] et par Wegner. Ce dernier exprime clairement sa motivation fondamentale : programmer des procédures interactives, c'est justement *renoncer* à vouloir décrire complètement le comportement du système, et admettre qu'on ne peut le décrire que selon différents points de vue :

> Abandonner l'objectif d'une spécification complète du comportement requiert certes un ajustement psychologique, mais il rend respectables les méthodes d'ingénierie logicielle de spécification partielle incomplète d'un système par ses interfaces. Puisqu'un éléphant complet ne peut être spécifié, l'accent est mis sur la spécification de ses parties et de ses formes de comportement (comme sa trompe ou sa façon de manger des cacahuètes). La spécification complète doit être remplacée par une spécification partielle des interfaces, des vues et des modes d'utilisation.
>
> — Agents de voyages : faire des réservations pour le compte de clients
> — Passagers : faire des réservations directes
> — Employés des compagnies aériennes : faire des demandes au nom des clients et vérifier leurs billets
> — Agents de bord : assister les passagers pendant le vol lui-même
> — […][39]

En d'autres termes, programmer un système interactif, ce n'est justement pas programmer leurs interactions elles-mêmes, c'est-à-dire *quand* A rencontrera B, ou *quand* C donnera D à E, mais seulement leur *possibilité*, que A *peut* rencontrer B, etc. – laissant ensuite ces possibilités advenir de manière globalement non-déterminée, à partir d'une situation initiale et d'une stratégie interne à chaque agent. Wegner dénomme ces possibilités d'interactions *interfaces*, en référence probable à la programmation objet*, où ce concept joue un rôle important. On re-

38. (Hewitt, Bishop et Steiger 1973, p. 238).
39. (Wegner 1997, p. 86).

trouve dans la position de Wegner la même radicalité fondationnelle que celles de Kay, Hewitt et Milner : pour lui toute programmation n'a trait qu'à des interfaces.

Ainsi, la programmation est aussi importante par ce qu'elle dit que par ce qu'elle ne dit pas. Elle créée un espace de processus que leurs interfaces dirigent de manière lâche. Wegner parle de *harnais* imposés au calcul :

> Les descriptions d'interface sont appelées harnais, car elles servent aussi bien à contraindre le comportement du système (comme le harnais d'un cheval) qu'à exploiter celui-ci à des fins utiles. Les harnais ont une connotation négative en tant que contraintes, et une positive en tant que spécifications de comportement utile[40].

Un changement de perspective important a lieu ici. Les procédures interactives, dont la spécification est *partielle*, selon l'expression de Wegner, ne visent pas à résoudre des problèmes qu'il est possible de spécifier entièrement à l'avance, puisque ceux-ci se précisent dynamiquement dans un dialogue constant entre l'investigateur et la situation. On a affaire plutôt ici à *la préparation d'environnements pour la composition dynamique de procédures*. On programme certes toujours des machines, mais on n'attend plus d'elles qu'elles résolvent des problèmes, comme le ferait une investigation classique ; on les dispose de manière à ce qu'elles rendent effectives un écosystème de règles effectives et disponibles pour collaborer à la résolution de problème.

On peut comprendre cela autrement encore, en revenant au concept de *spécification*. Dans notre première partie[41], nous l'avions caractérisé comme une *interface* entre la machine, qui est un objet dont on contrôle le comportement, et la situation dans laquelle se situe le problème à résoudre. À présent que nous avons caractérisé la machine comme un micro-espace de systématicité, nous voyons que les machines ne nous permettent pas seulement de concevoir des procédures qui déterminent les opérations d'un seul centre de contrôle, mais des spécifications qui régentent les interfaces entre entités actives, qu'il s'agisse d'objets physiques, de machines ou de personnes. Ces spécifications ne déterminent pas ce que doit être *un* programme, mais comment doit se comporter *toute la situation*.

On retrouve ici la notion de *système objectif*[42]. Dans un système objectif, comme une institution, ou un système technique, des règles symboliques sont associées aux objets eux-mêmes de l'interaction. Il y a par exemple l'interdiction d'avancer que me signifie un feu rouge, ou au contraire la possibilité d'intenter

40. (ibid., p. 86).
41. Voir plus haut, § 54.
42. Voir plus haut, § 183.

un procès que m'accorde le tribunal. Il y a également la possibilité que m'offre la plateforme logistique d'ordonner, par un simple coup de fil, l'envoi d'un camion de 10 tonnes vers tel endroit, à tel coût. Le système désigne donc ici des objets rendus capables de coopération par des procédures déjà définies. Un système objectif est donc à la fois un ensemble de règles et de procédures disponibles, qui ne m'indiquent pas quoi faire, mais qui disent que tout ce que je fais doit l'être *avec elles* – soit comme moyens, soit comme contraintes.

§ 228. Les espaces de travail

Le concept de programmation correspondant à un système objectif peut être décrit, en première approche, comme un *espace de travail* (*workspace*). Un catalogue de tâches possibles est proposé aux interlocuteurs, en général associés à des objets symboliques comme des documents de travail. Les interlocuteurs peuvent créer et modifier ces objets comme dans le flux de tâches, mais des séquences précises ne sont pas imposées. Par contre, des mécanismes sont mis en place pour gérer les modifications simultanées d'un même objet. Des droits distincts peuvent également être attribués aux utilisateurs (il y a en général un administrateur de l'espace de travail), et ceux-ci peuvent interagir entre eux par différents outils de communication. Ce concept est pratiquement aussi vieux que l'ordinateur, puisque les systèmes d'exploitation, comme Unix, ont été initialement conçus pour résoudre ce problème, avec cependant plus d'accent mis sur la séparation des espaces individuels (représentés par le fameux « bureau » (*desktop*)des interfaces graphiques modernes) que sur les espaces partagés. Ceux-ci, historiquement développés dans certains milieux professionnels, comme l'ingénierie, se démocratisent aujourd'hui grâce aux outils de collaboration génériques comme Microsoft Teams.

De manière plus abstraite, un réseau informatique est également un espace de travail, car son système d'exploitation crée, au-dessus des ordinateurs physiquement connectés, une couche d'abstraction qui référence les ressources partagées (documents, mais également imprimantes, accès à Internet, etc.) et les utilisateurs dans un *annuaire*, une application critique qui référence les droits individuels de ces derniers.

L'espace de travail est donc essentiellement une collection organisée et cohérente d'opérations disponibles et de registres (dans le sens défini à la section précédente) qui régissent les droits qu'ont des agents de réaliser ces opérations dans une situation donnée.

Dans tous les exemples ci-dessus (le système d'exploitation, le bureau, la plateforme de collaboration, le réseau informatique) l'espace de travail sert à organiser l'accès aux ressources de la machine. Cependant, tout comme le registre et le flux de tâches, ce schéma de système de règles peut être couplé à des réalités non informatiques.

Outre l'exemple du système de réservation aérienne auquel fait allusion Wegner dans la citation ci-dessus, qui fut l'un des premiers projets d'informatique civile massivement distribuée[43], les ERP*, que nous avons déjà rencontrés[44], peuvent être vus comme des espaces de travail couplés avec le système opérationnel d'une entreprise. Au sein de ces vastes logiciels sont organisées les règles associées à toutes les ressources critiques de l'organisation : factures, bons de commande, ordres de production, statuts de commandes, stocks, comptes de trésorerie, rapports opérationnels et financiers, etc. Ces règles permettent de localiser ces ressources juridiquement et physiquement (*« à qui la ressource appartient ? à qui est-elle confiée ? où se trouve-t-elle aujourd'hui ?... »*). Elles déterminent comment ces objets peuvent interagir avec les flux de tâches définis, quels agents peuvent accéder à ces objets et les modifier, etc. Un ERP ne fait que rendre effectives les règles des registres et des flux des tâches définis en son sein, si bien, comme nous l'avons vu, que ses propres processus s'entrelacent avec les mouvements réels des ressources et avec les activités réelles des agents. Ainsi un mouvement de stocks n'est plus possible s'il n'est pas validé d'abord dans le système informatique, où telle activité administrative dépend d'un calcul qui doit être réalisé par l'ordinateur.

Ces couplages systémiques peuvent concerner des systèmes cyber-physiques. Nous revenons ici à l'exemple prospectif, mais parlant, du véhicule autonome, en évoquant les possibles évolutions de la signalisation routière. Celle-ci est aujourd'hui composée de panneaux de « sens interdit », de feux rouges, de passages piétons, etc. Ces objets sont marqués de couleurs vives ou contrastées afin que les automobilistes les remarquent bien et qu'ils puissent les interpréter correctement pour conduire selon des règles stables et établies. Si le système de signalisations *en général* est un système de significations, il s'agit bien ici d'un système objectif, puisque ses éléments ont une existence dans l'expérience, persistante dans le temps, et qu'ils interagissent avec les automobilistes pour contraindre leurs choix.

43. (Lesuisse 2018).
44. Voir plus haut, § 51.

En préparation de l'arrivée des véhicules autonomes, de nombreux projets sont aujourd'hui en cours pour intégrer des émetteurs aux signalisations routières afin que celles-ci communiquent directement avec ces véhicules[45]. Nous avons vu en effet que la complexité majeure des projets de véhicules autonomes était de leur permettre d'interpréter leur environnement visuel et sonore. Or il est assez évident qu'il serait bien plus efficace que voitures et signalisations communiquent directement entre elles, plutôt que tenter d'apprendre aux caméras du véhicule à péniblement reconnaître des panneaux et à interpréter leur sens. Cela est peut-être indispensable concernant les obstacles naturels de la route, mais tous les objets techniques du système routier – marquages au sol, signalisations, feux rouges – pourraient se signaler aux voitures de manière non visuelle. On appelle cette technologie V2I (*vehicle-to-infrastructure communication*). Les panneaux de signalisation ne sont plus alors seulement des objets passifs, mais deviennent des processus concurrents et interactifs. On en a un exemple primitif avec nos parcmètres qui sont devenus ces dernières années des véritables *registres*, au sens défini plus haut, capables d'être consultés et rechargés à distance. Dans tous ces cas de figure, la règle est immédiatement effective, car le véhicule autonome obéit directement à la signalisation (sans interprétation et décision d'un conducteur humain), et le parcmètre (ou plutôt le système de stationnement) verbalise directement le véhicule stationné indûment. L'informatique donne ainsi aux règles institutionnelles une *effectivité* affranchie du bon vouloir des agents humains, elle en fait, d'une certaine manière, des règles techniques.

Les avantages identifiés dans le cas du badge d'accès, à la section précédente, s'appliquent donc pareillement, mais ici au niveau de la circulation routière tout entière : gain de productivité dans l'administration et la police des règles (par *police*, nous entendons ici d'une manière générale l'ensemble des dispositifs mis en place pour veiller à l'application des règles), traçabilité, nouvelles fonctionnalités possibles. C'est sur ce dernier point que nous aimerions insister ici. En effet, du moment que les signalisations routières peuvent directement interagir et « négocier » avec les véhicules, on peut imaginer par exemple, que les sens interdits deviennent programmables pour valoir seulement certains jours de marché, ou être autorisés aux riverains, ou à la police, ou à tout véhicule en situation d'urgence, etc. Les règles de circulation pourraient être ainsi être adaptées très finement à des situations précises, car leur exactitude et leur effectivité immédiate écartent le risque

45. Voir par exemple (Frost 2020).

d'une confusion humaine d'interprétation.

Il est entendu ici que nous ne statuons pas sur les risques techniques d'un tel dispositif très largement prospectif, ni sur sa désidérabilité sociale. L'exemple du véhicule autonome nous intéresse pour des raisons conceptuelles, car il est aisé à représenter et possède une double profondeur, à la fois technique et institutionnelle. Il permet d'illustrer comment un système institutionnel ou technique, grâce à son couplage avec un espace de travail programmable, peut se particulariser « à l'infini » tout en restant systématique.

La programmation n'est plus ici rattachée à la résolution d'un problème local, ni même d'une classe de problèmes, mais s'attelle à la *construction de systèmes* tout comme nous l'avions vue soutenir, au chapitre précédent, les investigations théoriques globales[46]. Simplement il ne s'agit pas ici d'établir et de vérifier des systèmes de *significations*, mais bien de construire, dans l'expérience, des systèmes objectifs interactifs, c'est-à-dire des réseaux d'objets capables d'interagir entre eux et avec la situation, selon des règles, afin que s'y composent des procédures de résolution de problème.

§ 229. Les réseaux de services

Un espace de travail est tourné vers des interlocuteurs humains, dans le sens qu'il met à leur disposition des tâches ou des flux de tâches dont ils restent les acteurs prépondérants.

Lorsqu'au contraire ce sont des agents robotisés qui interviennent majoritairement dans le progrès des flux de tâches, le rôle des interlocuteurs humains se limitant à les déclencher, à les superviser et à consommer (au sens large) leurs services, il est plus pertinent de parler d'un *réseau de services* interconnectés.

Pour l'illustrer de manière parlante, on peut prendre l'exemple d'une demande de passeport. Dans le cas simple, cette procédure administrative requiert la préparation d'un dossier comprenant l'ancienne pièce d'identité, un justificatif de domicile, une photo, et l'acquittement d'un timbre fiscal. Autrefois ces documents étaient des imprimés : une facture d'électricité, une photo produite par une cabine automatique, l'achat d'un timbre dans un tabac. Grâce aux initiatives de dématérialisation administrative, ce dossier est aujourd'hui constitué et transmis de manière électronique. Le justificatif de domicile est téléchargé par le requérant

46. Voir plus haut, § 218.

dans l'espace client de son fournisseur d'électricité. La cabine photo fournit un code sécurisé qui doit être transmis directement à l'administration, et le timbre fiscal est acquitté en ligne. Seule l'ancienne pièce d'identité reste matérielle, mais les initiatives d'identité numérique permettent déjà, dans certains pays, de passer outre cette condition.

On voit que, techniquement, ce flux de tâches pourrait être automatisé par l'interconnexion des services, que le requérant se contenterait d'autoriser, de la même manière qu'il fournit déjà le code de la photo et qu'il renseigne son code de carte bancaire pour régler le timbre fiscal. Tout simplement le reliquat historique des concepts de *timbre fiscal* et de *justificatif de domicile* n'ont plus d'utilité. Le premier correspond simplement à la preuve de réception d'un paiement par le Trésor Public, et le second à la garantie d'un tiers de confiance (le fournisseur d'électricité). La demande de passeport apparaîtrait alors comme un service de l'administration de l'État-Civil, qui sollicite d'autres services fournis par des tiers (la cabine photo, le fournisseur d'électricité, le Trésor Public, la banque) que le requérant n'a plus qu'à mettre en relation.

Ce principe d'interconnexion de services, encore prospectif dans l'administration, est déjà largement en place dans le cadre des transactions commerciales, ainsi que nous l'avons vu avec l'EDI*, qui permet aux entreprises contractantes de laisser les flux de tâches de leurs ERP* respectifs communiquer directement pour liquider la transaction, depuis l'émission du bon de commande jusqu'à réception des produits et acquittement de la facture[47].

En informatique, l'architecture de services n'est pas seulement un mode d'implémentation des programmes dans un réseau distribué. Il s'agit d'un principe de développement logiciel qui s'est progressivement imposé depuis trente ans, sous des appellations différentes[48]. Ces architectures, outre qu'elles ont contribué à l'émergence de l'informatique en nuages*, ont permis d'augmenter l'interopérabilité des applications et leur réactivité à leur environnement externe de plus en plus ouvert, mais également et surtout de rendre leur maintenance et leur évolution plus rapide. Le principe de modularité des fonctionnalités qu'elles promeuvent en effet permet de circonscrire les problèmes au maximum (comme par exemple une nouvelle fonction à programmer) et de permettre leur résolution par des tâches de

47. Voir plus haut, § 51 et § 53.
48. Voir plus haut, § 32. Pour une introduction aux architectures de service, voir (Fournier-Morel et al. 2020).

programmation indépendantes des services environnants.

Nous employons cependant le concept de *réseau de service* de manière plus large, pour dénoter tout ensemble d'applications interconnectées par des protocoles standardisés et collaborant de manière régulière ou sporadique pour la réalisation de tâches.

Des exemples de réseaux de services couplés à des institutions sont les places de marché et de manière plus générale tous les mécanismes financiers de compensation *clearing* où, aux agents humains plaçant des ordres d'achats et de vente, se substituent de plus en plus des automates exécutant des programmes prédéfinis, qui peuvent dépendre eux-mêmes de variables dynamiques, comme par exemple : *« Vendez telle valeur si son prix devient supérieur à tel montant ! »*, ou des règles bien plus complexes, comme *« Achetez telle valeur si l'indice du prix des polymères en Asie devient inférieur à tel montant, et que la banque centrale chinoise maintient ses taux inférieurs d'un demi-point aux taux américains ! »*, etc.

Ce principe est aussi à l'œuvre concernant la négociation de tâches entre des systèmes cyber-physiques. Par exemple, dans la technologie Kiva[49] de préparation des commandes en entrepôt, ce sont les armoires stockant les produits qui se déplacent vers l'opérateur préparant la commande, plutôt que le contraire. L'entrepôt est ainsi un gigantesque ballet d'armoires mobiles se coordonnant entre elles pour éviter les collisions et surtout les blocages, l'espace étant restreint (voir photo 12.2). L'opération complètement autonome de systèmes cyber-physiques coordonnés entre eux se développe dans un grand nombre d'espaces techniques, notamment les mines et les usines[50], où les personnes humaines sont concentrées sur des tâches de supervision et de résolution de problèmes.

On peut également distinguer espaces de travail et réseaux de services par la possibilité d'une gestion décentralisée des derniers. Un espace de travail, comme un ERP, est essentiellement normatif, donc rigide : il n'admet en son sein que les objets dont les types ont été définis par avance ; en contrepartie, il permet un partage de ressources efficace et maximal entre eux. Par contraste, les réseaux de services peuvent être décentralisés : seule est nécessaire la définition de protocoles d'interconnexion entre services, comme l'EDI, grâce auxquels chaque service peut échanger des données avec tout autre. Ils ne peuvent cependant rien *partager* à proprement parler, comme par exemple un document de travail ; ils doivent se

49. (Ma et Koenig 2017 ; Yudiansyah et al. 2020).
50. (Grabowska 2020 ; Stephenson 2023).

FIGURE 12.2 – La technologie logistique Kiva.
Source : Joel Eden Photography/Kiva Systems

l'échanger ou se donner explicitement des droits de modification à son propos. Une illustration du niveau de décentralisation possible des réseaux de services est le *contrat à exécution automatique* (le mal-nommé *smart contract*, « contrat intelligent »)[51]. Basé sur la technologie blockchain*, il permet à deux parties de conclure une transaction conditionnelle à la survenue d'événements futurs. Par exemple, l'achat d'une voiture de particulier à particulier pourrait être conclue par la réception d'un signal GPS que celle-ci se trouve bien au domicile de l'acquéreur suivie par l'exécution d'un virement vers le compte bancaire du vendeur. Ce sont donc des événements numériques, couplés à des événements du monde réel, qui constituent la transaction. L'intérêt de ces objets pour notre réflexion, c'est qu'ils promeuvent une conception *décentralisée* des transactions et des opérations en général : les règles de ces dernières n'ont plus apparemment besoin de cadre prédéfini pour devenir effectives. Ici, nul espace de travail, ni de système institutionnel, ni de système technique, ne semble requis. Il s'agit plutôt d'*interconnecter* deux flux de tâches afin qu'ils puissent synchroniser des conditions entre eux (par exemple le signal GPS devient condition du paiement), sans se donner la peine de

51. (GARAPON et LASSÈGUE 2018, p. 139-165) est à la fois une bonne introduction et une bonne analyse de ses implications sociales et politiques.

construire un espace de travail global comme le ferait un ERP. Il y a donc là une forme de retour à la « constellation » de significations dont parlait Dewey à propos du langage[52].

§ 230. La prolifération des règles

Les progrès importants de l'informatisation dans toutes les pratiques humaines ne veut donc pas dire que les machines en prendraient le contrôle, mais que se généralise l'exigence d'une exécution directement effective des règles – en conformité d'ailleurs avec notre première caractérisation des machines, qu'elles sont d'abord un idéal-limite de l'exécution humaine des règles[53].

On peut expliquer cette exigence par les différents avantages qu'on peut en retirer – productivité dans l'administration et la police des règles, nouvelles possibilités offertes quant à leur définition, exactitude de l'exécution, contrainte exercée sur la situation et traçabilité – ainsi que nous l'avons fait dans l'exemple du registre et du badge d'accès[54].

Il nous semble cependant y avoir également une logique interne à la généralisation de ce mode de régulation des pratiques humaines, qui dépasse les avantages qu'il apporte, et qui rend d'autant plus difficile sa limitation lorsque ses désavantages et ses risques, qui sont également nombreux, deviennent patents. Cette logique interne est celle de la systématisation elle-même.

Ce qui nous semble central ici est, parmi les avantages mentionnés, celui de pouvoir adapter les règles de manière indéfinie aux circonstances de la situation, ce qui revient à les rendre dynamiques, c'est-à-dire modifiables par leur environnement. L'informatisation n'est certes pas une condition nécessaire pour qu'un système objectif soit doté d'une certaine dynamique de ses règles. Ainsi, quand un « état d'urgence » est décrété, une institution bascule dans un autre régime de règles. Ou encore, pour reprendre l'exemple cité plus haut, la plateforme logistique peut être en rupture de stock, les camions retardés par une grève, etc. Des procédures de secours se mettent alors en route pour assurer les commandes, ou informer les clients de retards, etc.

Cependant le caractère dynamique des règles et des procédures reste nécessairement rudimentaire tant que l'informatique n'entre pas dans la partie. En effet,

52. Voir plus haut, § 170.
53. Voir plus haut, § 202.
54. Voir plus haut, § 224.

tant que l'effectivité des règles dépend de la sûreté des inférences que des agents humains peuvent faire, elles doivent être limitées en nombre et en complexité. Il est bien connu des dirigeants d'organisations que les livres de procédures interminables ne sont pas compris ni appliqués, et ont souvent l'effet inverse de leur finalité initiale : ils sont facteurs de désordre et d'arbitraire. Comme nous l'avons déjà dit plus haut, en l'absence de système informatique, la plupart des exceptions sont résolues hors d'un système écrit de règles, par négociation directe entre les intéressés. Si l'exception est régulière, la solution choisie devient une habitude tacite ou explicite entre eux, mais une habitude n'est pas encore une règle.

L'informatique change la donne, en permettant de superposer aisément et finement plusieurs régimes de règles, entre lesquels basculer selon la situation rencontrée. Dans les exemples donnés, la dynamique reste simple et à la portée d'un investigateur humain, qui doit simplement conditionner ses inférences à des conditions d'environnement complémentaires dont il doit d'abord s'enquérir (*« Est-on en état d'urgence ? Y a-t-il rupture de stock ? etc. »*). Mais l'informatique permet en droit de concevoir des règles de 3ème ordre qui peuvent encore modifier le système d'inférences, requérant à l'investigation de vérifier encore plus de conditions avant de savoir quelle règle appliquer. Cela est compliqué pour une intelligence humaine, mais un agent robotisé n'y éprouve aucune difficulté. L'informatique permet ainsi à nos systèmes d'inférence objectifs de prendre une profondeur potentiellement indéfinie. De plus, l'exploration de ces conditions n'adopte pas une structure nécessairement hiérarchique puisque des systèmes de règles dynamiques peuvent être composés de manière à avoir entre eux de multiples dépendances. On a alors affaire à un réseau dont le comportement global dépend de l'état de chacun de ses objets, à chaque instant.

La prolifération des règles est donc partiellement due à la souplesse que peut leur apporter un traitement automatique, puisqu'il est à présent possible de les affiner de manière indéfinie pour les rendre plus pertinentes.

Dans une situation où de tels régimes de règles effectives se développent à de multiples endroits, leur coordination par des personnes humaines devient rapidement difficile, ceux-ci ne pouvant s'adapter à la finesse des règles en jeu. À mesure que les participants à l'institution ou au système technique – aussi bien ses interlocuteurs externes (usagers, clients, administrés) que ses agents internes (techniciens, employés, fonctionnaires) – ne parviennent plus à appréhender aisément quelles règles doivent être appliquées dans la situation présente, voire qu'ils n'en

ont plus même connaissance, leur dépendance envers le système informatique pour conduire les opérations augmente. Pour reprendre l'exemple de la plateforme logistique, le système de routage des flux de marchandises peut être si complexe qu'il devient impossible de le réaliser manuellement. Lors d'une panne de son système informatique qui a duré plusieurs jours, une équipe logistique dut ainsi redéfinir des règles de routage simples pour les flux majoritaires, dégradant d'une manière drastique les exigences de délais et de coûts, et traiter toutes les exceptions manuellement[55].

Ici, le bénéfice de l'automatisation n'apparaît pas tant comme gain de productivité que comme *capacité* simple des humains à appliquer les règles. Aussi y a-t-il également besoin de *protéger* de tels systèmes d'interventions humaines intempestives. Toute interaction requiert une interprétation de la part de la personne, et est donc un facteur de risque. On ne peut non plus être sûr qu'elle aura les compétences nécessaires à cette interaction, étant donné la diversité des personnes. Enfin, l'interaction doit souvent prendre une forme sensible, coûteuse en ressources, et qui ajoute des ambiguïtés à la situation.

L'essentiel est donc d'éviter la rupture potentielle de l'intégrité d'un système. Apparaissent alors comme particulièrement malencontreuses toutes les situations où l'humain est en interface entre deux systèmes – où l'interconnexion des systèmes n'est pas encore réalisée. Par exemple, il peut s'agir pour l'opérateur humain de valider des données issues d'un système avant de les transmettre à un autre, ou de prendre une décision dans le cours d'un flux de tâches par ailleurs robotisé. De telles tâches sont parfois curieusement difficiles à automatiser ; ce n'est pas alors un critère de gain de productivité qui entre en jeu, mais tout simplement celui de garantie d'intégrité du système. Une tendance importante de l'informatique de gestion depuis quelques années est ainsi le développement de logiciels spécialisés dans l'automatisation d'interfaces entre applications, appelée communément automatisation robotisée des processus[*56].

On peut se représenter cette exigence d'interconnexion de manière graphique de la façon indiquée dans la figure 12.3, qui reprend les principes du schéma 3.11 en représentant les facteurs propres de la situation considérée en vert clair (par exemple, les agents humains de transactions commerciales), en vert foncé les règles d'interface entre la situation et la machine (la spécification de ce que doit réaliser

55. Expérience personnelle de l'auteur.
56. (Gerbert et al. 2017 ; Hofmann, Samp et Urbach 2020).

la machine dans la situation, par exemple exécuter un ordre de vente) et en bleu les règles de comportement de la machine, dans son propre référentiel, celui de son langage de programmation. Dans le graphique de gauche, une situation relative-

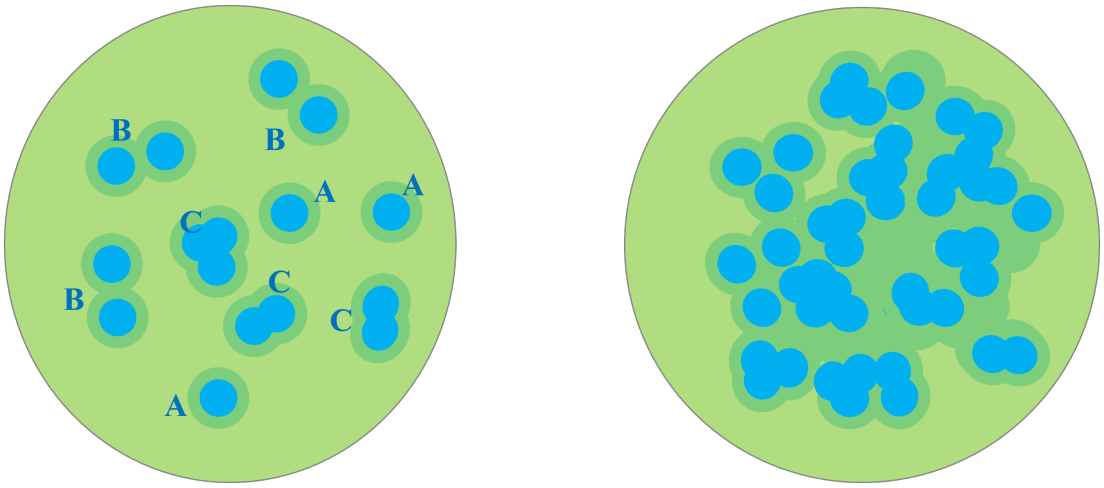


FIGURE 12.3 – Représentation du phénomène d'interconnexion des systèmes

ment peu systématique est représentée. De nombreuses machines informatiques, labellisés « A » contribuent au fonctionnement de la situation de manière isolée. Les systèmes « B » sont composés de plusieurs machines qui communiquent entre elles par le biais de protocoles de communication, comme l'EDI*, ce qu'on a représenté par la jonction de leur cercle extérieur vert foncé, mais non de leur cercle intérieur bleu. Enfin, les systèmes « C » possèdent plusieurs machines fonctionnant de manière intégrée, c'est-à-dire partageant plusieurs modules informatiques directement.

Le graphique de droite montre une situation orientée largement automatisée, où les machines se sont largement intégrées les unes aux autres pour laisser le moins de place possible à une coordination externe ; il n'y a pratiquement plus qu'un seul grand système de type « C » autour duquel gravitent quelques systèmes « B ». Cela ne veut pas dire que les facteurs propres de la situation (notamment ses agents humains) ne jouent plus aucun rôle et ont disparu – mais ils sont relégués à la périphérie du système interactif.

Cet entrelacement conduit aujourd'hui à un *couplage* de fait entre le système technique ou l'institution (la mine, l'usine, la plateforme logistique, l'entreprise, l'administration, le marché, la communauté, etc.) et son système d'information. Ce couplage n'est rien d'autre que celui de deux systèmes de règles – ou plutôt

encore, la prise en main d'un seul système de règles, celui de l'institution, par une compétence de programmation. Ce couplage est réalisé aujourd'hui à des degrés divers. Nous avons évoqué son importance dans les entreprises commerciales et les administrations publiques. Il est presque entièrement achevé concernant le système bancaire et les systèmes de communication. Concernant le système bancaire, les actifs et les flux financiers sont à présent attestés, sinon définis, par leur traçabilité dans des systèmes informatiques. Le reliquat qui leur échappe – l'argent liquide, les chèques – est en forte diminution. Il est intéressant de remarquer que les nouveaux actifs financiers qui échappent au système bancaire – les monnaies numériques, comme le BitCoin – sont entièrement définis par des procédures informatiques. Quant aux systèmes de communication actuels, ils ont déjà conduit une bonne partie de l'humanité à créer des profils numériques avec lesquels ils sont étroitement couplés, et auxquels ils s'identifient partiellement. Dans le domaine des infrastructures (routes, réseaux, etc.) par contre, les évolutions sont plus lentes, du fait du coût souvent prohibitif qu'il y a à insérer de l'électronique dans des systèmes techniques matériels déjà existants.

§ 231. La désintermédiation

Nous appelons *désintermédiation* le phénomène décrit au développement précédent, où l'interconnexion des services relègue les agents humains à la périphérie de la situation. De ce fait, les raisons pour lesquelles le système central prend telle ou telle décision peuvent devenir opaques pour les agents humains de la situation, ceux-ci ne pouvant appréhender, ni l'ensemble des paramètres pris en compte, ni le programme complexe qui les traite. Il est utile ici, pour les mêmes raisons et avec les mêmes mises en garde, de prolonger encore une fois l'exemple prospectif du véhicule autonome.

Dans une ville où seuls de tels véhicules seraient autorisés, la complexité mentionnée plus haut pour les machines d'interpréter un environnement fondamentalement humain *disparaîtrait*. La route serait en effet conçue d'emblée *pour les machines*. Il n'y aurait plus alors de marquage au sol, ni de signalisation. *De facto*, il n'y aurait plus même de règles fixes, puisque celles-ci pourraient être rendues flexibles afin de mieux atteindre les objectifs premiers du code de la route, comme la sécurité des passagers et la fluidité du trafic. Il n'y aurait plus ainsi de sens interdits, mais seulement des règles de priorité, des feux s'adaptant à la congestion du trafic, des véhicules s'adaptant leur vitesse et même leur itinéraire les uns aux

autres, etc. Dans un tel environnement, *ce serait une intervention humaine qui serait dangereuse*.

Ce n'est pas seulement l'action humaine qui serait désintermédiée ici, mais également le mode visuel de la communication humaine, très inefficace pour des machines. Le panneau lui-même, avec ses marques colorées, deviendrait inutile et disparaîtrait, et avec lui, l'explication pour le passager de la décision de son véhicule. Sans doute cela ne ferait-il pas grande différence, tant la conduite autonome, dans ce scénario prospectif, diffèrerait sans doute de la conduite humaine. Bien plus que le mode de communication, c'est le mode de raisonnement humain qui serait également désintermédié. La règle à appliquer se trouvant être le résultat, chaque fois particulier, d'un programme complexe tenant compte de l'ensemble de la circulation urbaine, perdrait ainsi ses qualités de généralité et de stabilité qui permettent aux personnes humaines de raisonner à son propos.

Nous touchons ici un point fondamental que rencontre toute investigation qui se place dans le cadre de systèmes objectifs interactifs : les manières humaines de faire les choses lui sont des obstacles plutôt que des moyens. Marx l'avait déjà perçu, à la suite d'Ure qu'il cite :

> « La faiblesse de la nature humaine », s'écrie l'ami Ure, « est si grande que plus l'ouvrier est qualifié, plus il sait ce qu'il veut et plus il est difficile à manier, et que par suite ses sautes d'humeur causent un grand préjudice au mécanisme global[57]. »

Il ne s'agit pas ici de caractéristiques morphologiques, mais bien de principes différents d'action – le mécanisme dans la citation de Marx désigne la division du travail. La logique de l'action des machines diffère de celle des personnes. On retrouve ici l'idée promue par Wagner (1998), de faire droit à la notion de *pensée mécanique* qui désignerait les compétences logiques propres des machines, et qu'il s'agirait d'étudier en tant que telle, c'est-à-dire indépendamment de sa comparaison avec les compétences propres des personnes humaines :

> On ne cherche pas à prouver que les phénomènes psychiques peuvent être expliqués par une analyse fonctionnelle ou un programme informatique ; on s'efforce de mesurer quelles sont les opérations qu'il est possible de mécaniser et quelles sont celles qui résistent à toute tentative de formalisation et de programmation. On ne postule aucun recouvrement des capacités de l'esprit par celles des machines, aucune identité de nature entre la pensée mécanique et la pensée d'un être humain ; on définit seulement l'idée d'une pensée non humaine afin d'en comprendre les limites et les

57. (Marx [1867] 2006, p. 414).

> propriétés. La notion de pensée mécanique n'est donc pas mise au service d'une philosophie de l'esprit ; elle doit permettre de constituer une philosophie des machines. À l'intelligence artificielle orientée par le projet d'une programmation des capacités de l'esprit humain s'oppose une pensée mécanique définie, du point de vue empirique, par un ensemble de performances relatives au raisonnement, au langage, à l'action, aux connaissances, et fondées sur des possibilités de calcul et de traitement de l'information[58].

Nous pensons en particulier que c'est dans le domaine de l'action et de l'interaction qu'il est le plus important de dégager la perspective des machines (c'est-à-dire des systèmes de règles dynamiques contrôlant des objets de l'expérience), afin de comprendre quels modes de coopérations sont possibles avec les personnes.

Dans cette perspective, beaucoup d'efforts actuels réalisés dans l'intelligence artificielle, d'apprendre aux machines à « penser comme des humains », peuvent sembler transitoires, et témoigner d'un développement intermédiaire de la logique technologique. La conception de véhicules autonomes serait simplifiée radicalement s'il s'agissait de les faire circuler sur des voies dédiées et protégées. Le problème se poserait alors de manière fondamentalement différente, puisqu'on ne s'intéresserait pratiquement plus à obtenir et à traiter des informations visuelles de l'environnement du véhicule, mais seulement à bien définir les standards de description et d'échange d'information entre infrastructures et véhicules. L'irréalisme de tels projets ne tient pas à leur logique mais, d'une part, au coût de transformation des infrastructures routières, une autre forme de dette technique, et d'autre part à leur acceptabilité sociale.

Cette tendance à la désintermédiation complète de l'action humaine dans les systèmes interactifs est un autre facteur d'opacité épistémique. Ces systèmes semblent ne plus être conçus *pour nous* car notre incapacité à suivre leurs ratiocinations les conduit à nous repousser à leurs frontières, c'est-à-dire à des positions de purs consommateurs, requérant des services à l'entrée et les consommant à la sortie.

12.4 La programmation continue

Les deux sections précédentes ont tenté de mettre en évidence différents schémas de la programmation interactive, par lesquels se réalisent des couplages entre

58. (Wagner 1998, p. 204).

des objets de l'expérience humaine et des systèmes de règles dynamiques. On voit que toutes ces formes sont complémentaires : les réseaux de services et les espaces de travail sont constitués de registres et de flux de tâches, mais ces derniers ne donnent leur pleine mesure que lorsqu'ils s'exécutent dans les cadres posés par les premiers.

La programmation de machines, dans ce cadre, devient naturellement *programmation de systèmes*, c'est-à-dire préparation d'environnements rendant possibles des processus interactifs et les contraignant à la fois, définissant ainsi des espaces de chemins d'interaction. Il y a ici une analogie stricte avec le rapport que nous avions établi entre investigation théorique (globale) et résolution de problèmes (locale). Programmer une machine, c'est résoudre un problème dans une situation donnée. Programmer un système, c'est établir une « théorie » de la situation, qui passe certes par des résolutions locales de problèmes (des programmations particulières), mais qui sont organisées en fonction de la visée architectonique globale.

Nous venons d'affirmer qu'une telle programmation systématique tendait a reléguer les personnes à la périphérie du système, et à les rendre simple consommateurs des services rendus. On pourrait objecter ici qu'une telle situation concerne l'utilisateur, et non le programmeur du système qui, en étant l'architecte, a au contraire (idéalement) une position d'omniscience sur l'ensemble des services développés en son sein.

Nous voudrions dans cette dernière section avancer qu'une telle objection n'est que partiellement vraie. Dans un réseau de services indépendants interconnectés, le programmeur n'a de vue que sur les services qu'il a lui-même développés ; quant à l'architecte, il ne fait que fournir des services d'infrastructure qui permettent de mettre en relation les autres services, mais qui ne déterminent ni leur contenu, ni leur comportement. Surtout, une telle programmation a elle-même tendance à devenir interactive, adaptant de plus en plus rapidement le système aux besoins de la situation : le programmeur devient un élément du système global, au même titre que l'utilisateur, le maître d'ouvrage, et le réseau environnant de services eux-mêmes évolutifs.

Nous commençons par analyser la transformation de la pratique même de la programmation lorsque celle-ci a pour objet un réseau de services, c'est-à-dire une infrastructure logicielle persistante dont elle doit assurer l'évolution et le bon fonctionnement (§ 232). Il apparaît alors que la construction d'infrastructures

interactives est à la fois une accumulation – ce qui permet à chaque programmeur de disposer de tout ce qui a été développé dans le système jusqu'à lui – et une dette – car il dépend aussi d'un historique qu'il ne maîtrise plus (§ 233). Une situation globale d'insécurité réglementaire apparaît alors – les paramètres de celles-ci étant dépendantes d'un état global du système, lui-même en constante évolution (§ 234).

§ 232. La programmation continue

Dans les architectures de service, la programmation est tout entière tournée vers l'insertion de petits composants applicatifs dans une infrastructure logicielle existante. D'une certaine manière, le programmeur se trouve dans la même configuration qu'un informaticien de réseau qui réalise différentes tâches sur celui-ci : ajouter ou supprimer des utilisateurs, intégrer de nouveaux composants comme des ordinateurs ou des imprimantes, construire de nouvelles interfaces, etc. Il n'y a plus de distinction stricte entre le temps du développement et le temps des opérations*, puisqu'à tout moment un programmeur peut ajouter de nouveaux composants au système ou les mettre à jour. Il y a certes toujours une procédure stricte de mise en production*, mais celle-ci, au moins théoriquement, est beaucoup plus courte que dans les architectures logicielles classiques.

Afin de développer cette idée, il nous semble utile de l'observer dans toute sa pureté conceptuelle, telle qu'elle apparaît au sein du langage visionnaire SmallTalk (1972). Son concepteur, Alan Kay, y décrit l'idée d'*objet* comme celle d'un petit ordinateur placé dans un réseau d'autres ordinateurs similaires[59], ou comme une récurrence sur l'idée même d'ordinateur[60].

Dans un tel contexte, la distinction entre utilisateur et programmeur s'efface. Le premier interagit avec le système en envoyant un message à un objet, et en observant sa réaction conséquente, ou plutôt celle du système tout entier. Le second, on s'en rappelle, construit de nouveaux objets en définissant ses réactions possibles à des messages, réactions qui sont elles-mêmes constituées de modifications de ses objets internes et d'envois de messages à d'autres objets. Or cette construction se fait elle-même par l'interaction du programmeur avec des objets déjà actifs dans l'espace de travail Smalltalk.

En effet, la définition d'une classe* d'objets se fait toujours en envoyant des messages à d'autres classes, qui sont elles-mêmes des objets pour Smalltalk. Dans

59. (Kay et Ram 2003).
60. (Kay 1996, p. 513).

des langages plus traditionnels, une classe est souvent comparée à une *espèce* d'objets, au sens de la logique classique. Ici, la bonne métaphore est plutôt celle de l'usine. Programmer une classe, c'est construire une usine, et l'instancier, c'est demander à l'usine de produire un élément[61].

Il n'y a plus, concrètement, de distinction forte entre l'écriture du programme, sa compilation et son exécution. Toute nouvelle définition de classe est immédiatement exécutée et devient un objet disponible dans l'environnement (une « usine »). La distinction entre programmeur et utilisateur, quant à elle, réside simplement dans le fait que le premier interagit avec certains objets (les classes) plutôt qu'avec d'autres (leurs instances), mais une telle distinction n'a plus de frontière nette. Tous deux interagissent dans le même espace de travail, et rien n'empêche l'utilisateur de définir ses propres classes à tout moment.

Ce style incite à l'écriture de programmes en petites unités, dont le code fait en général quelques lignes, et qui se composent entre elles par instanciation, par modularité (un objet peut être partie d'un autre objet), par interaction (un objet peut envoyer un message à un autre objet le faisant ainsi réagir) et deux ou trois autres relations que nous ne détaillons pas ici. Chaque application Smalltalk peut être ainsi décrite comme un réseau d'objets modulaires interagissant entre eux. SmallTalk invite ainsi à programmer de manière expérimentale, par la capacité qu'il donne à interagir directement avec des objets en cours de construction.

Ce n'est donc plus la métaphore de l'artefact qui est pertinente ici, mais bien celle du jugement, au sens de Dewey et de Martin-Löf[62] qui est simplement précédée par la délibération (l'écriture du programme). Nous avions dit[63] que décider d'exécuter un programme, c'est juger que sa spécification est conforme aux besoins de la situation. Ici, toute programmation est immédiatement jugement, et l'investigation est bien, conformément à la thèse de Dewey, la construction d'un jugement complexe à partir de tels jugements partiels ; seulement tout se déroule à un niveau supérieur de l'investigation traditionnelle, qui manipule en général des significations statiques. Une bonne comparaison serait la délibération du stratège militaire qui, commandant ses troupes sur le champ de bataille, sait que celles-ci ne sont pas de simples pions, mais des entités actives qui vont par elles-mêmes modifier la situation.

61. Voir la remarque sur le tutoriel du langage Squeak, la version actuelle principale de SmallTalk, sur squeak wiki.
62. Voir plus haut § 174 et § 178.
63. Voir plus haut, § 193.

Smalltalk est un exemple précurseur de la programmation de systèmes, qui consiste essentiellement à insérer de nouveaux composants dans un écosystème de composants actifs. La mise en production* d'une application nouvellement développée, à partir d'un certain niveau de complexité, est de ce type. Elle requiert la mobilisation de ressources informatiques diverses (serveurs, disques, réseaux), à différents niveaux d'abstraction. Il faut établir des connexions entre elles, déclarer des utilisateurs, créer les droits d'accès de ceux-ci à ces ressources, etc. Toutes ces étapes complexes d'affectation de ressources* sont la charge d'informaticiens spécialisés. Par ailleurs, au-delà de l'environnement matériel, des connexions avec d'autres systèmes ou services, locaux ou distants, doivent en général également être établies (par exemple l'accès à des bases de données, à des services Internet), afin qu'ils puissent interagir avec eux. Cela exige des protocoles souvent complexes de traduction de données, comme on l'a vu dans l'exemple de l'EDI dans l'industrie ophtalmique[64]. Il s'agit là d'une programmation éminemment interactive, puisqu'il s'agit, à proprement parler, de déclarer à des objets *déjà actifs* de nouveaux objets dont ils doivent tenir compte selon les règles spécifiées.

Dans une architecture de services, la mise en production est simplifiée par la standardisation des environnements et des procédures, mais c'est parce qu'elle devient le geste essentiel du programmeur. Très souvent, il ne s'agit pas de spécifier ou de coder, car le service qu'il s'agit de mettre en place est un logiciel standardisé. Le geste essentiel est ce qu'on appelle l'intégration – qui implique aussi bien la paramétrisation du service que sa connexion aux bonnes bases de données et aux annuaires d'utilisateurs.

Ces nouvelles exigences modifient en profondeur le travail de développement, en le rendant beaucoup plus proche de ce qu'on appelle les opérations* en informatique, c'est-à-dire les tâches liées à la maintenance et à l'exploitation du système d'information dans sa globalité, tant au niveau de ses infrastructures que de ses fonctionnalités. L'approche Devops*, populaire depuis une dizaine d'années, reflète cette réorientation, et est souvent associée aux architectures de services[65]. Elle favorise l'intégration continue*, c'est-à-dire la mise en production régulière de petites modifications apportées au logiciel, ce qui est la réponse opérationnelle aux méthodes de développement agile*.

Or la difficulté de cette insertion n'est pas seulement, ni même principalement,

64. Voir plus haut, § 53.
65. Voir plus haut § 32, (Fournier-Morel et al. 2020, p. 10-11).

informatique. Dans les exemples d'évolutions de besoin évoquées plus haut[66], de nouvelles règles venaient contredire les précédentes – comme par exemple celle qui donnait l'ordre à la production de mettre de côté les commandes du vendredi venait contredire la règle générale de produire les commandes aussi vite que possible. La difficulté se situe ici *au niveau des règles elles-mêmes* et du conflit entre spécifications, par exemple entre règles de sécurité et règles d'accès, entre règles de qualité et règles de flexiblité, etc. La programmation n'est plus ici centrée sur les machines et sur les objets, mais sur la situation tout entière qu'il faut organiser de manière systématique ; c'est-à-dire concevoir, non plus les règles de comportement effectif d'un objet, mais les règles d'interactions possibles.

La complexité de la programmation de systèmes ne réside plus tant dans la résolution d'un problème donné – car des solutions existent pour la plupart des besoins standardisés d'une organisation, que dans l'adaptation d'une solution existante à son écosystème, dans la *négociation* de l'insertion réussie d'une nouvelle règle active dans un réseau de règles actives – sans que cela entraîne d'incompatibilité (c'est-à-dire d'incohérence) ou de failles de cas d'usage (c'est-à-dire de défaut de systématicité).

Dans cette perspective, il semble possible de distinguer trois niveaux logiques fondamentaux (ce qu'on appelle aussi les paradigmes de programmation*) dans la programmation :

1. La programmation élémentaire est fonctionnelle, au sens que les processus programmés n'interagissent pas avec leur environnement et se contentent de transformer des entrées en sortie. Son modèle est le λ-calcul*.
2. Un second niveau de programmation est interactif, au sens que l'état de la machine programmée peut être modifiée, en cours d'exécution du processus, par l'environnement. On appelle souvent à tort ce paradigme *impératif*, mais sa caractéristique fondamentale réside dans le fait que la mémoire du processus peut être accessible à d'autres processus ; il inclut donc la programmation concurrente. Son modèle est le π-calcul*.
3. Le troisième niveau, que nous avons esquissé ici, est celui de la programmation *de contraintes*, ou encore non-déterministe. Ici il ne s'agit pas de programmer un processus, mais des systèmes de règles qui se contentent de *contraindre* tout processus devant être exécuté sous son égide. Un tel système est non-déterministe car il ne dit pas aux processus ce qu'il faut faire,

66. Voir plus haut, § 83.

mais seulement ce qu'ils ne doivent pas faire. Il ne s'agit pas, par exemple, de programmer un algorithme de sortie de labyrinthe, mais le labyrinthe lui-même, de sorte que son « exécution » donne *tous les chemins possibles* pour en sortir. Un tel paradigme nous semble (dans l'état de nos connaissances) encore imparfaitement dégagé. Prolog est le langage modèle d'une telle programmation, cependant son orientation logique ne permet pas de traiter l'interactivité des processus au sein des systèmes programmés. Comme nous l'avons vu, des systèmes d'objets actifs, comme ceux de Smalltalk, mais qui se contenteraient de *contrôler* ce qui se passe dans la situation, et non de l'animer, semblent assez proches de ce qui est ici visé.

§ 233. L'accumulation et la dette réglementaire

Dans le paradigme du développement de systèmes que nous venons d'esquisser, programmer consiste essentiellement *à faire advenir de nouveaux objets* dans un environnement dit *de production*, c'est-à-dire dans lequel les utilisateurs interagissent *déjà* avec le système d'information. On pourrait dire, d'une certaine manière, qu'il s'agit d'une programmation *située* qui doit s'accommoder de la situation dont elle hérite et qui ne peut la modifier que de manière incrémentale – comme le navire en haute mer d'Otto Neurath[67], qu'on doit réparer par parties car il doit continuer de naviguer. Le programmeur ne peut repartir de zéro, car même s'il voulait réécrire entièrement le code de son système, il devrait encore s'adapter à toutes les ressources dont il dépend ou qui le sollicitent (bases de données, applications tierces, utilisateurs, etc.). Cela veut aussi dire qu'il ne peut jamais être entièrement responsable du comportement de son service, puisque celui-ci dépend de tous les autres services avec lesquels il interagit ; il est tributaire de sa bonne analyse de cette situation et de ces dépendances.

Cela se voit de manière impressionnante dans les applications informatiques complexes, comme les systèmes d'exploitation, construits par des milliers de programmeurs, dont chaque nouvelle contribution vient s'ajouter, ou corriger, ou étendre les contributions antérieures. On ne parle pas ici des propriétés cumulatrices du savoir, qui sont si bien connues, mais concrètement, d'objets qui s'ajoutent et s'empilent les uns sur les autres, comme les étages d'un immeuble en construction. Par exemple, lorsqu'on installe un paquetage* applicatif sur Linux, le gestionnaire d'installation vérifie ses dépendances envers d'autres paquetages,

67. (Neurath 1973, p. 199).

et si ceux-ci ne sont pas installés, il procède de manière récursive. Ces dépendances sont déterminées par le fait que des procédures du paquetage cible convoquent pour leur exécution des fonctions définies dans les paquetages requis, ce qui veut dire encore que les premières ont été construites sur la base des secondes. La figure 12.4 montre les dépendances du compilateur **gcc**, dont on

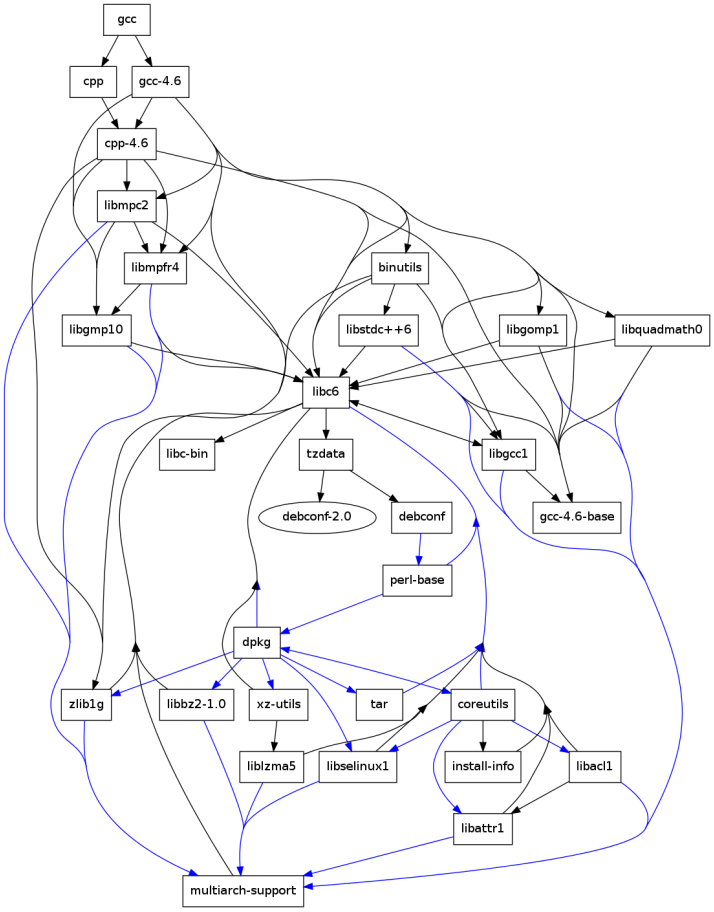


FIGURE 12.4 – Les dépendances du paquetage **gcc**, obtenues par la fonction **apt-redpends** sur **Ubuntu**.
Source : www.ubuntubuzz.com

voit qu'elles cascadent pour beaucoup sur plus d'une dizaine de niveaux (sans compter les dépendances internes de fonctions à l'intérieur d'un même paquetage) : cela illustre l'ampleur de ce phénomène de construction itérative à l'œuvre dans la programmation.

Cette propriété existe en une moindre mesure dans d'autres techniques, par

exemple dans l'ingénierie mécanique, qui construit des objets complexes à partir de sous-ensembles techniques, ceux-ci étant construits à partir de pièces détachées. Mais cette construction itérative n'exhibe jamais la flexibilité illustrée par notre figure. Cela tient à ce que *construction* ici ne doit pas être entendue comme un « ajout de matière », mais comme la modification de circuits de transmission de manière à y faire apparaître des structures, enchâssées à plusieurs niveaux en nombre indéfini. Ce qui est programmé, encore une fois, sont des règles qui agissent sur d'autres règles, ou coopèrent entre elles, et c'est cette dimension récursive possible de l'activation de règles par d'autres règles qui crée la possibilité d'une construction itérative.

La dimension positive de cette propriété est assez évidente. Elle permet à chaque programmeur de bénéficier de tout le travail accompli jusqu'à lui. Elle explique la sophistication impressionnante de nos ordinateurs et des dizaines de couches logiques qui coopèrent entre elles pour réaliser les opérations de haut niveau que nous leur demandons.

La dimension négative est moins souvent aperçue, sauf par les informaticiens eux-mêmes. On peut l'appeler la *dette informatique* (anglais : *technological debt*, bien que *legacy* soit le terme le plus usité pour évoquer ce problème). Possède une telle dette tout programme qui dépend de programmes antérieurs, écrits parfois il y a plusieurs années, par des auteurs devenus anonymes, à la documentation clairsemée, dans des langages peu usités. Si par exemple une erreur survient dans ces programmes requis, ou si une modification doit leur être apportée, ou si le système qui les héberge vient à défaillir, tous les programmes qui en dépendent sont en danger de devenir eux-mêmes défaillants ou non fiables. Aussi, des budgets conséquents sont dépensés par les organisations tous les ans pour maintenir en bonne condition leur système d'information, typiquement, en installant les dernières mises à jour de leurs logiciels, en démantelant de vieux systèmes ou de vieilles technologies, en vérifiant le code de leurs programmes, l'intégrité de leurs bases de données, etc.

Or, du fait de la convergence entre programmation et conception, on peut craindre que ce phénomène de prolifération et de dette réglementaire s'étende à tous les objets couplés avec des systèmes de règles dynamiques, étant donné la facilité avec laquelle la programmation interactive permet d'ajouter des règles à un système interactif, ou de moduler celles qui existent par d'autres. Une étude pourrait être faite concernant les liens entre l'informatisation de nos sociétés et

la prolifération des règles et des codes qui régentent nos artefacts et nos institutions. On peut prendre l'exemple de l'interaction entre comptabilité et fiscalité : étant entendu aujourd'hui que les objets comptables, par exemple les factures, existent avant tout numériquement, le droit fiscal s'autorise à exiger une traçabilité toujours plus grande à leur égard – par exemple, en demandant une preuve de livraison – ce qui peut obliger certaines entreprises à intégrer leur système comptable à leur système logistique de manière plus forte qu'elles n'auraient souhaité le faire de leur propre chef. En ingénierie civile, nous avons noté que le BIM* permet de mieux vérifier la conformité réglementaire d'un bâtiment : aussi y a-t-il de nombreuses initiatives pour traduire les réglementations de construction en langage BIM[68]. Lorsque les autorités de régulation auront établi de tels standards, une tendance similaire à celle qu'on observe concernant la fiscalité des entreprises peut être anticipée.

Cependant, le risque d'une dette réglementaire est la sclérose. En particulier, dans les investigations de conception, si les contraintes édictées sur les espaces de problème sont trop élevées, ceux-ci pourraient devenir trop difficiles à reconfigurer, notamment parce qu'ils seraient trop dépendants de la manière dont les programmes les représentent. Ces contraintes de représentations pourraient empêcher la créativité humaine de reformuler les problèmes, c'est-à-dire de les poser en de tout autres termes, – ce qu'on sait être une étape majeure de l'invention.

D'un point de vue épistémologique, la dette informatique, qui est au fond une dette réglementaire, est une autre cause d'opacité épistémique. La nature collaborative et cumulative de la construction de systèmes de règles fait que chaque programmeur n'en connaît qu'une partie limitée, quand bien même il en serait l'architecte principal.

§ 234. L'insécurité

Une autre propriété épistémique de la programmation de systèmes objectifs interactifs est qu'il n'y a plus qu'une distinction de degré entre utilisateur et programmeur, comme nous l'avons relevé à propos de SmallTalk.

On peut voir un ordinateur comme une famille d'espaces de travail. Pour un programmeur, il est un environnement de développement. Pour l'utilisateur simple, il est par exemple un traitement de texte, puis un tableur, puis un jeu, etc.

68. (Preidel et Borrmann 2018).

L'intérêt de cette formulation est que la distinction entre programmeur et utilisateur y apparaît comme celle entre deux formes d'interactivité, différentes seulement relativement, et non absolument. Comme le rappelle Moor (1978), tout programmeur est l'utilisateur d'un programme écrit par un autre. Dans la profondeur indéfinie d'interactivité que présente tout ordinateur, on peut se placer sur un plan plus ou moins avancé, mais il n'y a jamais un arrière-plan absolu, car même les circuits imprimés de la machine, qu'on pourrait supposer jouer ce rôle, ont été programmés à partir d'autres machines. En ce sens, même l'utilisateur le plus inexpérimenté peut être dit composer une procédure, donc programmer en un sens très large. On retrouve ici les constats que nous avions fait au début de cette enquête, en particulier cette idée de Mélès (2022), qui l'appuie sur le principe fondamental de la relativité de la distinction entre programme et donnée. Entrer une donnée – comme écrire « a » sur son traitement de texte – c'est toujours envoyer, d'une manière ou d'une autre, une instruction à l'ordinateur.

L'aspect positif de cette convertibilité entre utilisateur et programmeur, entre donnée et programme, réside dans la convergence que nous avons mentionnée à propos de SmallTalk. D'une part, l'informatique met à disposition des utilisateurs des moyens pour contrôler le comportement de systèmes ; d'autre part, la programmation se simplifie pour ressembler à une simple utilisation intuitive de machines[69].

Son aspect négatif réside dans l'*insécurité* radicale dans laquelle se trouve tout utilisateur ou programmeur face à son système. L'insécurité s'entend d'abord ici dans le sens courant que le terme prend en informatique, où il connote l'intrusion non autorisée et malveillante d'agents tiers dans des systèmes. Cette intrusion peut avoir pour finalité le vol de données et d'identité. Mais nous nous intéressons particulièrement ici à l'autre forme que prennent ces actions, à savoir la modification, le blocage, voire la destruction du système compromis. La cause fondamentale de cette seconde insécurité est qu'un système objectif interactif a un *état persistent*, qu'il modifie en fonction de ses interactions avec le reste de la situation. Cette dernière étant en général ouverte, un interlocuteur donné ne peut jamais être vraiment sûr de l'état dans lequel se trouve son système. Le rôle des espaces de travail, déjà mentionnés à de nombreuses reprises, est justement de cloisonner les états du système accessibles à chaque utilisateur, et de sécuriser leur accès par diverses technologies de cryptage. Une autre expérience de cette incertitude fondamentale,

69. Voir plus haut, § 38.

que beaucoup ont déjà connue, est l'obligation de réinitialiser un système « sur ses données d'usines », selon l'expression consacrée, lorsque des modifications malencontreuses ont irrémédiablement compromis son intégrité.

Ces phénomènes ont également lieu au niveau des systèmes objectifs couplés à des processus interactifs. Certes, l'entrelacement de l'informatique et des systèmes techniques ou institutionnels réels, si prégnant aujourd'hui, passe en général inaperçu. Il se manifeste au public lors d'incidents critiques où des bogues informatiques paralysent des infrastructures majeures. Ce fait est si frappant qu'il est souvent magnifié dans les intrigues des « films-catastrophes » où il s'agit régulièrement de sauver l'humanité d'un système informatique trop zélé à accomplir ses tâches.

Tous ces phénomènes sont des conséquences du fait qu'un système objectif interactif est une chose *de l'expérience*, à laquelle accèdent des agents (qu'il s'agisse de personnes humaines ou d'autres systèmes externes) de manière concurrente pour collaborer à des procédures. Ces interlocuteurs peuvent avoir des droits d'accès différents aux ressources du système, mais il n'y a pas de différence fondamentale entre eux. Un utilisateur peut malencontreusement bloquer un système, si celui-ci a été mal conçu par un programmeur, – car tous, par leur interaction, modifient le système – c'est d'ailleurs pour garder trace de ces interactions qu'il a en général un état.

L'idéal d'une programmation fonctionnelle* pure a été (et est encore) poursuivie dans le but de remédier à cet état de fait, et de strictement encadrer l'interactivité. Un programme fonctionnel simple est non interactif, et ne fait que restituer à l'utilisateur des résultats que celui-ci sollicite en évaluant la fonction avec des arguments d'entrée. Un tel programme ne modifie pas l'état du système. Si une interaction est requise, la fonction qui convoque la ressource externe (entrée au clavier, lecture d'un fichier, connexion réseau) se termine à cette convocation et génère une autre fonction qui réagira à ces données. L'état du système est censé ainsi toujours rester stable et transparent. Cet esprit radical a animé la conception du langage Haskell, dont l'un des manuels les plus populaires s'explique ainsi, avec une note d'humour :

> En Haskell, une fonction ne peut pas modifier un état. […] La seule chose qu'une fonction peut faire dans Haskell est de nous renvoyer un résultat basé sur les paramètres que nous lui avons donnés. Si une fonction est appelée deux fois avec les mêmes paramètres, elle doit retourner le même résultat. Bien que cela puisse vous sembler un peu limitatif si vous venez d'un univers impératif [*NdT : impératif*

> *ici s'oppose à fonctionnel, et équivaut à notre* interactif], nous avons vu que c'est en fait vraiment sympa. Dans un langage impératif, vous n'avez aucune garantie qu'une fonction simple qui devait simplement calculer quelques chiffres ne brûlera pas votre maison, ne kidnappera pas votre chien et n'égratignera pas votre voiture avec une pomme de terre tout en calculant[70].

Ces exemples loufoques expriment bien le fait fondamental qui caractérise les systèmes objectifs interactifs, à savoir l'incertitude radicale dans laquelle on se trouve face à eux. Un autre terme possible serait en anglais *brittleness* – fragilité, ou friabilité, qu'on peut opposer à la *robustesse* exigée des théories scientifiques.

Il est important de comprendre que la programmation fonctionnelle ne remédie pas à cet état de fait – car l'état du système est modifié à chaque nouvelle fonction publiée dans l'environnement. Elle est seulement une méthode qui permet de mieux tracer et localiser ces modifications. En effet, l'insécurité, l'incertitude, ou la fragilité dont nous parlons ici n'ont qu'une seule origine fondamentale. Il s'agit d'un phénomène d'opacité épistémique, qui est l'ignorance de l'*état* complet du système à un instant donné. Cela ne doit pas être compris comme notre incapacité à appréhender d'un seul coup d'œil le contenu exhaustif de la mémoire d'un ordinateur. Après tout, cela serait faisable dans un symbolisme approprié. Le problème épistémique que nous avons est plutôt de dérouler toutes les conséquences de ce contenu, c'est-à-dire de suivre le système *dans l'évolution dynamique de ses règles*. Modifier l'état du système peut être « dangereux » parce que chaque nouvelle donnée pointe peut-être vers *une nouvelle règle*. Notre limitation épistémique fondamentale est que nous savons mener des investigations sur la base de systèmes de significations stables, alors que les règles d'un système objectif interactif interagissent entre elles en réseau. Ainsi, même s'il n'y a pas d'interactivité externe, mais que nous avons affaire à un système massivement concurrent de procédures internes, nous nous retrouvons en situation d'opacité au bout de quelques itérations, car ses règles se sont reconfigurées de telle sorte que nous ne savons plus lesquelles appliquer pour continuer à suivre son évolution. Certes, comme nous l'avons vu plus haut, les méthodes formelles peuvent indirectement nous permettre de démontrer certaines propriétés de ces systèmes. Nous éprouvons cependant comme une limitation de ne pouvoir reconstituer la procédure effective qui mène à un résultat. L'opacité de l'apprentissage-machine en intelligence artificielle tient à ce fait, qu'un système objectif interactif, massivement concurrent, fait évoluer ses règles rapidement en

70. (Lipovača 2011, chp. « Input and Output »).

fonction d'un grand nombre de données d'apprentissage[71].

Ici nous pouvons revenir à la critique, évoquée plus haut[72], que certains informaticiens pourraient faire à notre perspective « mathématique » sur la programmation, à savoir qu'elle ne peut rendre compte de l'esprit de « bricolage » et du pragmatisme qui règne *de facto* dans les projets de programmation. On se rappelle à cet égard les critiques de De Millo, Lipton et Perlis (1979) à l'égard des méthodes formelles[73]. À cette critique, nous pouvons répondre que cet esprit et cet état de fait – qui a néanmoins largement décru suite à la professionnalisation des méthodes de développement – n'est absolument pas incompatible avec notre perspective. Les programmeurs travaillent sur des systèmes instables par nature. Cette instabilité n'est pas due à la mauvaise qualité de leurs composants matériels, mais au fait que l'état de ces systèmes interactifs est déterminé par de larges enchevêtrements de règles internes et externes. Les efforts des architectes pour modulariser et compartimenter ces règles sont immenses, mais comme elles doivent, pour une large part, coopérer pour fixer l'état instantané de la machine, ils n'en peuvent garantir la fiabilité absolue.

§ 235. Conclusion

Comme nous l'avons vu au chapitre précédent, le premier domaine dont la programmation accroît l'effectivité est *l'investigation elle-même*, non seulement parce qu'elle accélère ses phases de ratiocination, mais également parce qu'elle est la modalité exacte par laquelle doit s'exprimer idéalement une investigation systématique – par des procédures définies exactement et exhaustivement. Les ingénieurs, les architectes, les scientifiques et les mathématiciens, les industriels et les logisticiens – et peut-être aussi un jour les juristes – conçoivent leurs ouvrages et leurs plans en programmant des ordinateurs, mais ils n'ont pas l'impression de programmer, car ils ne font qu'explorer, à l'aide d'ordinateurs, la systématicité de leurs constructions – leur exactitude et leur exhaustivité s'il s'agit de résolutions de problèmes, leur cohérence et leur richesse s'il s'agit de théories. Programmer est la manière naturelle pour ce faire, car la systématisation et la mise en procédure de l'investigation sont les deux facettes d'une même attitude de contrôle des inférences.

71. (Stephanou 2018).
72. Voir plus haut, section 11.3.
73. Voir plus haut, § 48.

Le second domaine dont la programmation accroît l'effectivité est l'ensemble de ce que nous avons appelé les *pratiques* humaines, en tant qu'elles sont encadrées par des institutions et des techniques. La programmation permet de rendre directement effectives des règles institutionnelles et techniques au sein de ces pratiques, modifiant par là profondément leur équilibre interne et leur dynamique. Notamment, les systèmes cyber-physiques et les couplages d'institutions réelles à des systèmes de règles dynamiques permettent de donner à ces dernières une flexibilité quasi-infinie, dont nous ne pouvions avoir l'idée avant l'avènement des ordinateurs.

Or il doit apparaître à présent que ces deux formes d'effectivité sont en réalité une seule. Il s'agit toujours de rendre des règles directement effectives dans une situation en y insérant des machines, que cette situation soit mathématique, scientifique, institutionnelle, technique, juridique, interpersonnelle, etc. Dans certains cas les machines réalisent des inférences qui transforment une situation « intellectuelle », dans d'autres elles modifient des éléments dans une situation « matérielle », mais ces distinctions ontologiques n'ont aucune importance ici, car les situations sont toujours, en tous les cas, des significations par lesquelles on décrit l'expérience. Dans les deux cas pareillement, la programmation ne se contente pas de résoudre des problèmes, mais participe à l'effort de systématisation qui la rend possible – qu'il s'agisse de systèmes théoriques dans le cas des sciences et des mathématiques, de systèmes techniques et institutionnels dans le cas des pratiques et de l'ingénierie. Dans les deux cas enfin, les avantages liés à l'automatisation (productivité, exactitude, nouveauté), s'appliquent pareillement : ils sont des conséquences directes de cet unique principe, d'utiliser une machine pour rendre une règle directement effective dans une situation. Il faut noter par contre que les deux autres formes d'avantages de l'informatisation, ceux liés à la contrainte exercée sur la situation et ceux liés à la traçabilité de l'interaction, semblent bien spécifiques aux situations d'interactivité pratique, et être moins pertinents pour l'investigation.

L'idée de *rendre une règle directement effective dans une situation* ne tombe pas de nulle part, et n'est pas survenue au détour d'un hasard technique. Elle est rendue possible par la résolution de l'ambiguïté lancinante concernant la fonction des règles tout au long des Temps modernes – *guider l'esprit* pour le rendre *plus présent* à la situation problématique en cours et lui permettre de mieux juger, ou se substituer à l'investigation par sa récapitulation procédurale, et *congédier l'esprit*.

Avec la révolution industrielle, cette ambiguïté est, dans le cas des problèmes pratiques, clairement résolue en faveur de cette seconde option[74]. Les règles ne s'opposent plus à la *routine*, mais la déterminent. On les énonce *après* l'investigation pour résumer celle-ci en une procédure qui permet de la remplacer par une simple ratiocination. L'exécution d'une telle ratiocination *exige* de l'esprit qu'il se comporte « mécaniquement », la machine étant le lieu logique des inférences certaines qu'on pouvait faire à propos de certains phénomènes naturels et des artefacts humains. Ici, les mécanismes de transmission ont joué un rôle aussi bien historique que paradigmatique, et leur évolution qui passe par l'horlogerie, les machines à calculer, la machine de Babbage, les machines à tabuler, etc. est sous-tendue par une recherche à la fois obstinée et tâtonnante, de rendre des règles directement effectives, par la transposition de leur système de signification sous-jacent dans celui d'un mécanisme de transmission.

On comprend à présent pourquoi une définition mathématique de la programmation[75] peut escamoter si aisément la référence à des machines : c'est que la machine concrète, si elle est le moyen irremplaçable d'une telle effectivité, ne sert que de noyau à la construction de machines logiques, institutionnelles, techniques, etc. bien plus complexes, dont les règles s'expriment directement dans le système de significations de la situation elle-même. Ces machines, en tant que telles, n'apparaissent pas nécessairement comme machines (sauf dans les environnements techniques) car elles sont avant tout des systèmes de règles dynamiques effectives, et c'est bien leur description mathématique, c'est-à-dire purement procédurale, qui apparaît comme légitime, c'est-à-dire comme la plus conforme à leur intention. Ce qui importe, c'est de résoudre des problèmes dans une situation donnée, ou plutôt même encore, d'organiser la situation de manière à ce que le problème n'y apparaisse plus : c'est là que l'idée de règles directement effectives s'impose, et par là le recours aux machines comme chevilles ouvrières de cette effectivité. Mais on voit ici que, une fois résolus les problèmes de transcription, ces machines ont vocation à disparaître de l'horizon de l'investigation, celle-ci revenant à la manipulation des règles qui l'intéressent et à leur modalité d'effectuation. La programmation devient si spécialisée à chaque domaine pratique qu'elle n'apparaît plus comme programmation – ainsi les exemples de Catala, des tableurs*, etc.

Nous avons tenté enfin de montrer, dans ce dernier chapitre, que ce mouve-

74. Voir plus haut, section 7.2.
75. Voir plus haut, section 11.1.

ment de rendre les règles effectives s'auto-alimentait. Les règles font système, et ces systèmes, lorsqu'ils sont interactifs, ne se laissent pas en général juxtaposer aisément. Il y a une sorte de logique naturelle à laisser les règles « s'interconnecter », d'autant qu'elles deviennent plus nombreuses et plus complexes. Ce faisant, nous avons essayé de mettre en évidence les risques épistémiques majeurs qui se développent, ces systèmes devenant de plus en plus opaques aux personnes humaines et donc incontrôlables par elles : invisibilisation des machines qui rendent possible ces systèmes interactifs, désintermédiation des personnes dans la résolution des problèmes, dette et insécurité réglementaires.

Pour illustrer d'une autre manière encore ces phénomènes d'opacité, on peut reprendre l'image des livres exhaustifs de procédures auxquels l'intelligence humaine est si rétive[76]. Une étape nouvelle est franchie dans la découverte qu'*il est possible de concevoir des procédures interactives*, soit encore, selon les mots de Wegner, des *spécifications partielles*, qui soutiennent l'investigation et l'action réfléchie, sans pour autant la prédéterminer à l'avance. Du moment en effet que l'environnement interagit avec la procédure, cette image du livre se brouille. On imagine qu'il faudrait que chaque procédure identifie, à chaque étape, toutes les possibilités de réponse de l'environnement, comme le font les livres-aventures pour enfants. Cependant il n'y a dans ces ouvrages qu'une interactivité statique et limitée. La réalité est que l'état de la situation et chaque objet en son sein peuvent influencer chacune des règles à venir de la procédure. Pour conserver la métaphore du livre, il faudrait en imaginer un qui, en fonction de la situation, nous dirait quoi faire et quel autre livre chercher – ou encore mieux, comment recomposer un nouveau livre pour affronter les étapes suivantes.

Le paradoxe auquel aboutit ici l'entreprise de systématisation est donc qu'elle entraîne, par ces phénomènes d'opacité, une perte de contrôle de la personne humaine sur le monde qu'elle habite, c'est-à-dire *la perte de ce qu'elle cherchait justement à assurer* – et nous n'avons encore rien dit des déséquilibres de pouvoir et d'opportunité entre les personnes que ce nouveau désordre peut engendrer.

76. Voir plus haut, § 71 et section 5.1.

Chapitre 13

Conclusion

Notre conclusion a d'abord une vocation récapitulative. Notre première section (13.1) vise à vérifier que nous avons répondu aux nombreuses questions ou étonnements qui ont émergé au cours de ce travail. Cette synthèse met en lumière deux questions qui restent encore à traiter.

La première question traite de la portée de l'apprentissage machine statistique qui, depuis une dizaine d'années, et plus particulièrement encore depuis quelques mois, remet l'intelligence artificielle au cœur du débat public, et bouleverse les pratiques de programmation, jusqu'à la prédiction de sa disparition. Nous tentons une première approche de ce phénomène *à partir* des concepts que nous avons développés dans ce travail (section 13.2).

La seconde question est plus fondamentale et porte sur la nature et le sens de l'effectivité des machines. Nous avançons qu'elle ne doit pas être comprise comme procédant de l'énergie qui les alimente ou de la robustesse des matériaux qui les constituent, mais comme celle d'un *détour productif* qu'entreprend le sujet connaissant et agissant afin d'augmenter l'effectivité originaire de la connaissance, selon le seul principe de la *cohérence de ses intentions* (section 13.3).

Dans la dernière section 13.4, nous ouvrons notre réflexion aux débats éthiques et politiques que soulève la prolifération des systèmes de règles interactifs, étudiée au chapitre précédent.

13.1 Reprise d'ensemble de la problématique

Nous rappelons d'abord (§ 236) les nombreuses questions ou étonnements qui ont émergé lors de notre parcours, notamment dans l'introduction et les deux premières parties, avant de leur apporter aux deux développements suivants (§ 237, § 238) des réponses succinctes.

Ces conclusions ne sont pas exhaustives ; nous avons renoncé à énumérer ici les nombreux thèmes de philosophie des sciences, des techniques, de la logique et des mathématiques auxquels nous espérons que ce travail pourra contribuer.

§ 236. Reprise des questions de ce travail

Notre étonnement initial portait sur la présence généralisée des ordinateurs dans notre monde humain (§ 1)[1] et sur la difficulté que nous avions à nommer la fonction générique de ces machines (comme le transport pour les voitures, la communication pour les téléphones, etc., § 2). Nous avons alors posé les trois hypothèses suivantes (§ 11) :

(1) L'ordinateur est une machine polyvalente parce qu'elle est programmable.
(2) La programmation est une activité qui radicalise une forme de connaissance rationnelle.
(3) La forme de connaissance qu'est la programmation doit expliquer comment et en quel sens des machines peuvent être *effectives*, c'est-à-dire contribuer à la résolution de situations humaines problématiques.

La première hypothèse entraîne toute une série de questions assez élémentaires concernant la programmabilité des machines et la nature de celles-ci. D'abord, nous sommes-nous demandé, comment est-il possible qu'un ensemble de formules symboliques, ou une suite de 0 et de 1 – ce à quoi se réduit ultimement le programme d'un ordinateur – puisse s'animer et s'exécuter de manière à accomplir des fonctions signifiantes dans le monde, comme piloter un avion, ou même simplement déterminer l'affichage d'un écran ? Comment est-il possible d'établir des correspondances entre des significations et des phénomènes matériels, ou encore : comment des signes peuvent-ils déterminer le comportement d'une machine (§ 39) ? et que faut-il, d'ailleurs, entendre par ce dernier terme : *machine* ? Faut-il le prendre dans le sens restreint des ordinateurs de Von Neumann*, ou l'étendre à

1. Nous associons chaque question au numéro du développement ou de la section où elle est apparue, qui servira à l'identifier aux deux développements suivants.

tout type de machine, au sens courant du terme, voire encore aux machines métaphoriques et aux machines abstraites que recense également le dictionnaire (§ 40, § 59) ? Enfin, cette question peut également prendre une tournure inverse : quelles sont les conditions pertinentes pour affirmer qu'un système physique « calcule » (question de la *physical computation* dans la philosophie anglo-saxonne, § 22) ?

L'hypothèse (2) entraîne une longue série de questions concernant la relation entre programmation et connaissance. Peut-on dire qu'elle est une technique, ou une mathématique ? Il semble qu'on puisse continument la décrire comme abstraite ou concrète, théorique ou pratique (§ 12). Ou est-elle plutôt une *nouvelle* forme de connaissance – voire une science générale de second degré, destinée à améliorer toutes nos autres facultés de connaissance – comme le suggère le mouvement de la *pensée computationelle* (§ 10) ou, comme l'affirme de manière plus radicale Herbert Simon avec son idée d'une science de la *conception* (§ 70) ?

Ces dernières hypothèses, qui font de la programmation une activité *générique* de connaissance, peuvent éclairer les difficultés qu'il y a à définir, voire à tout simplement *cerner* ce qu'est un programme ou un algorithme (§ 21). L'activité de programmation en effet se fragmente en une multiplicité de formes et de langages (§ 22), se transforme sans cesse et se rend invisible (section 2.1). Notamment, elle peut se simplifier à l'extrême et se mettre à la portée de jeunes enfants, ou s'intégrer sans discontinuité aux méthodes de travail des professions, comme l'architecture, si bien qu'il est difficile de la cerner dans les statistiques économiques (§ 27, section 2.4). Il n'est pas même possible de distinguer, sinon par degrés, la figure du programmeur de celle de l'utilisateur (§ 43), ni celle de l'amateur du professionnel. Enfin, nous avons noté la surprise des programmeurs face à la *difficulté* de la programmation, et leur assertion répétée que les machines *raisonnent différemment* que nous[2].

Toutes ces difficultés s'éclaireraient si la programmation était, comme le soutient Simon, la forme générale de la raison pratique : son caractère protéiforme ne serait alors rien d'autre que celui de l'agent connaissant, occupé à transformer le monde. Cependant, une telle hypothèse se heurte à au moins deux difficultés : d'une part, si la programmation est située dans une telle proximité de la connaissance, pourquoi ne réussit-on à dégager son objet positif dans toute sa généralité (la procédure) que si tardivement, à l'occasion de l'invention d'un artefact technique contingent, l'ordinateur (§ 13) ? D'autre part, comment répondre aux thèses

2. Voir plus haut, § 10, § 29.

qui se répandent aujourd'hui, à peine 75 ans après son apparition, selon laquelle elle est obsolète et vouée à être remplacée par l'apprentissage machine statistique (§ 25) ?

Enfin, notre hypothèse (3) porte sur la nature de l'*effectivité* des machines. Nous avons noté, au début de notre questionnement, que ce terme pouvait se prendre en un sens large – celui de *rendre effectif, réel* une intention, et en un sens informatique restreint, plus récent, celui des *procédures effectives* : la question de l'articulation de ces deux significations est restée ouverte (§ 3). En § 14 et § 19, nous avons montré l'inscription de ce questionnement dans une problématique philosophique plus large, qui porte sur l'*effectivité* de la connaissance en général, qui a fixé deux questions directrices à notre recherche épistémologique. D'une part, quel rapport entretient la connaissance de *ce qu'il faut faire* pour résoudre tel ou tel problème avec la connaissance simple de *ce qui est* ? En est-elle dérivée (une « application »), ou au contraire est-elle un mode autonome de connaissance, qui se constitue selon des principes propres ? D'autre part, comment des procédures pourraient-elle augmenter notre capacité de connaître elle-même – ainsi que le suggère l'importance que prend l'informatique en sciences et en mathématiques ? Cela signifie-t-il que la connaissance a le caractère d'un *faire*, d'une pratique susceptible de prescriptions et de techniques qui peuvent en augmenter l'effectivité ?

Ces deux questions montrent la proximité de notre sujet à de nombreuses autres problématiques philosophiques contemporaines, qui se sont d'ailleurs grandement trouvées reconfigurées par l'apparition du phénomène informatique. Nous avons cité la question de l'autonomie des techniques vis-à-vis des sciences (ou le contraire, si on suit le paradigme aujourd'hui quasi-dominant de la *technoscience*, § 15), celle des logiques de l'action et des formes de vie (§ 16), de la cognition étendue (§ 17), et de la dualité épistémique des règles (§ 18)[3].

Enfin, nous nous sommes interrogés sur la *puissance* de l'informatique (§ 20), question qui ouvre notre sujet aux très nombreux débats éthiques et politiques qui traitent de ses conséquences dans notre monde humain.

§ 237. Réponses aux questions (1)

Nous répondons ici de manière succincte à la plupart de ces questions, sans les reprendre explicitement afin de ne pas alourdir le texte et permettre une lec-

3. Les règles « procrustéennes » et « canoniques » de Ryle. Voir également l'opposition des règles et des routines en § 114.

ture continue. Nous indiquons seulement, par une note de bas de page au début de chaque paragraphe, les numéros des développements ou sections associés aux questions traitées.

Nous commençons par les questions liées à notre hypothèse centrale (2).

Les deux questions directrices de notre réflexion épistémologique[4] ont été l'objet de nos trois parties centrales ; nous en avons présenté une synthèse en section 10.6, à laquelle nous renvoyons la lectrice.

Cette réflexion a notamment permis de dégager la nature et la possibilité d'une *résolution systématique de problème*. La programmation est une de ses formes[5], ce qui la rapproche des mathématiques, des sciences, de l'ingénierie et de l'industrie. Elle se distingue de celles-ci en ce qu'elle a affaire spécifiquement à la mobilisation de machines pour effectuer cette résolution ; c'est à ce titre qu'elle *accompagne* toutes ces disciplines, ainsi que toutes les pratiques humaines qui présentent des problèmes où une telle forme de résolution apparaît bénéfique. L'informatique, en tant qu'elle est une *science des opérations*, qui développe et étudie des systèmes de règles dynamiques (*i.e.* représentant et transformant des situations), est ainsi une mathématique de second ordre, comme la théorie de la preuve. Elle se développe librement en théories particulières (les langages de programmation) selon les classes de problèmes traitées[6].

C'est ce caractère très générique de la programmation qui explique la polyvalence des ordinateurs et les difficultés qu'il y a à cerner leur fonction[7]. La programmation peut prendre autant de formes que les règles par lesquelles nous tentons de régenter ou transmettre nos pratiques, qu'elles soient celles du jeune enfant qui s'exerce à mettre en place une interaction réglée avec son monde environnant, ou celles du professionnel qui cherche à résoudre des problèmes sophistiqués dans son propre domaine d'expertise. En particulier, il est vrai qu'il n'y a pas de distinction de droit, mais seulement de degré entre programmeur et utilisateur. L'utilisateur sollicite en effet l'ordinateur dans le cadre d'une investigation (sauf s'il joue), et à ce titre lui fournit des séquences d'instructions, donc le « programme »[8]. Cependant ces séquences ne sont pas en général récapitulées en une procédure systématique pour une réutilisation future ; il s'agit là, selon nous, de la différence majeure qu'on

4. Questions § 14 et § 19.
5. Réponse à la question § 12.
6. Voir plus haut, § 196.
7. Réponses aux questions § 1 ,§ 2, § 21, § 22, § 27, § 43, sections 2.1 et 2.4.
8. Voir plus haut, § 234.

peut trouver entre ces deux modes d'interaction avec la machine.

Il faut cependant se garder d'identifier cette activité avec la forme pure et universelle de la raison, comme le fait Simon et, dans une moindre mesure, le mouvement de la *pensée computationnelle*[9]. Nous avons montré que la posture systématique dans la résolution de problème constituait en une large mesure une rupture épistémique avec les postures pratique et technique habituelles ; elle entre souvent en conflit avec ceux-ci, et elle est loin d'être toujours d'une pertinence *supérieure*[10] ; cela est principalement le cas dans les problèmes de très grande échelle, où les questions de productivité, d'exactitude et d'exhaustivité deviennent prépondérantes[11]. Par ailleurs, nous avons plusieurs fois noté que par *résolution systématique de problème* il ne fallait pas entendre que l'investigation procédait *elle-même* systématiquement (sauf dans des cas très précis), mais que son résultat avait les vertus de la systématicité[12]. Au contraire, atteindre la systématicité est intrinsèquement difficile pour les personnes humaines, car cela suppose de mettre à distance nos écologies cognitives habituelles, ainsi que les concepts et les inférences qu'elles nous suggèrent, afin de les critiquer et de les vérifier de manière ratiocinative, une posture hautement *artificielle* pour notre intelligence[13].

Comment émerge cette rupture épistémique[14] ? Cette question est complexe, car elle touche à celle de l'historicité des formes de pensée. Il nous a semblé que la résolution systématique de problème émergeait difficilement, et à tâtons, lors des Temps modernes européens. Nous consacrons le développement § 242 à cette question. La question complémentaire de la disparition possible de la programmation, telle que nous la connaissons, au profit de l'apprentissage machine statistique, est l'objet de la section 13.2.

Nous abordons à présent le groupe de questions liées à l'hypothèse (1), concernant la programmabilité des machines.

La possibilité qu'il y a de confier à des machines l'exécution automatique de procédures systématiques[15] requiert d'élucider *ce qu'est* une machine : un phénomène matériel dynamique dont nous pensons pouvoir décrire l'interaction avec

9. Réponses aux questions § 10, § 70.
10. Voir plus haut, section 5.4 et 5.2.
11. Voir plus haut, section 5.3 et 11.6.
12. Voir notamment § 29, § 77 et § 191.
13. Voir plus haut, § 80.
14. Questions § 13 et § 25.
15. Réponses aux questions § 40, § 59.

son environnement selon des règles[16] ; dès lors que nous pouvons, de plus, contrôler une partie de ces interactions, c'est-à-dire changer certaines conditions d'environnement qui déterminent le comportement de la machine, une forme minimale de programmation peut se mettre en place, que nous avons illustrée avec les mécanismes de transmission[17]. Une machine est donc un ensemble de règles que l'on réfère à un phénomène dynamique ; c'est parce qu'elle est une catégorie de significations qu'elle présente cette ambiguïté essentielle, de pouvoir être considérée comme un objet concret ou comme une abstraction. En tant qu'elle est un ensemble de règles, elle peut être identifiée au *concept* de tels phénomènes ; réciproquement tout concept peut être décrit comme une machine inférentielle, dont la dynamique est donnée par l'acte de juger de l'investigateur : c'est la machine abstraite des logiciens[18]. En tant qu'elle désigne un objet concret dans l'expérience, il se peut qu'on trouve des écarts entre le phénomène réel et les règles qui lui sont attribuées : c'est la machine concrète des techniciens. Il est licite d'étendre ce vocable de *machine* à toute sorte de phénomènes qu'on pense pouvoir régenter de manière stricte : ainsi une ferme, une administration, un ballet ou un orchestre, etc. : le caractère métaphorique de telles expressions est inversement proportionnel au degré de systématicité de leurs objets.

Dans cette acception très générale, toute construction ou modification de machine pour résoudre un problème peut être qualifiée de programmation[19] ; et *de facto*, nous avons noté la tendance de l'ingénierie mécanique à s'informatiser[20]. Si cependant une telle portée est jugée trop large relativement aux usages courants de ces termes, on peut restreindre la programmation au domaine des machines universelles, c'est-à-dire celles dont l'espace de systématicité est assez riche pour qu'on puisse y représenter le système de l'arithmétique[21]. La possibilité d'*« animer des 0 et des 1 »* afin qu'ils accomplissent des tâches signifiantes complexes dans le monde tient à ce qu'ils représentent le code établissant *la correspondance entre les règles* du système de significations cible (celui où est résolu le problème) et celles du système de significations de la machine.

16. Voir plus haut, § 204.
17. Voir plus haut, § 205.
18. Voir plus haut, § 142.
19. Réponses à la question § 39.
20. Voir plus haut, § 208.
21. Voir plus haut, § 205.

Dans ce cadre, la question du calcul physique (*physical computation*)[22] tient sa complexité à la seule ambiguïté de la notion de *calcul*. Dans l'acception que nous avons promue dans ce travail, le calcul est une forme de ratiocination systématique ; elle se réfère donc à l'exécution d'une procédure établie suite à une investigation, afin d'éviter la réitération de cette dernière. Lorsqu'on cherche, comme Piccinini, à en faire une notion de plein droit, indépendante de l'investigation, qu'il faudrait identifier aux mécanismes descriptibles par la seule modification des *degrés de liberté* caractéristiques du phénomène, on ne fait qu'affirmer d'une autre manière qu'on a affaire à une machine dont les règles seraient seulement procédurales, c'est-à-dire indépendantes des significations matérielles qui permettent de référer son concept à l'expérience[23]. Les paradigmes de telles machines sont bien sûr les contrôles de transmission.

§ 238. Réponses aux questions (2)

Pour terminer cette récapitulation, nous revenons à notre hypothèse (3) concernant le sens et la raison de l'*effectivité* de la connaissance et es machines, ainsi qu'à l'apport possible de notre travail à des problématiques philosophiques adjacentes.

Nous savons à présent que l'effectivité de la connaissance est un fait épistémique général et essentiel, car ses matériaux élémentaires, les significations, ne sont rien d'autre que des *règles* ; et une règle non effective, c'est-à-dire dont l'opération, sous les conditions indiquées, n'aboutit pas aux conséquences escomptées, est vite abandonnée ou réformée. Il n'y a rien de spécifique sur ce point à la résolution systématique de problème et à la programmation. Ce qu'apporte cette dernière est *un surcroît d'effectivité* de la connaissance grâce aux machines[24]. L'automatisation apporte des avantages relatifs de productivité, d'accès à de nouvelles opérations et d'exactitude concernant l'exécution des procédures. Mais la machine exerce également des contraintes sur son environnement, qui doit coopérer avec elle selon ses propres termes afin que la procédure réussisse ; enfin la machine permet la traçabilité des interactions et des opérations réalisées, ce qui offre à l'investigation une réflexivité nouvelle sur le déroulement des ratiocinations[25].

Dans notre dernière partie[26], nous avons vu que tous ces avantages s'ancraient

22. Réponse à la question § 22.
23. Voir plus haut, § 212.
24. Réponse à la question § 2.
25. Voir plus haut, section 5.4.
26. Réponse à la question § 1.

dans une qualité essentielle des machines, celle de pouvoir *rendre des règles effectives dans une situation donnée*, que celle-ci soit institutionnelle ou technique, ou encore (en étendant quelque peu la portée de *situation*) qu'elle soit simplement « intellectuelle », *i.e.* qu'il s'agisse de la situation de l'investigation elle-même. C'est cette qualité fondamentale qui, selon nous, explique la présence généralisée des ordinateurs dans notre monde humain et, tout aussi bien, leur invisibilisation : car ce sont avec les règles elles-mêmes que nous avons le sentiment d'interagir, et non avec les machines qui les rendent effectives.

Cependant, il n'est pas sûr que nous ayons par là répondu à notre hypothèse (3), bien au contraire. La qualité que nous prêtons aux machines, de *rendre les règles effectives*, est-elle bien élucidée ? Car quelle est la source d'effectivité des machines elles-mêmes ? On serait tenté de répondre, en première intention, que c'est leur source d'énergie ou de mouvement, par exemple le flux de l'eau, soumis à la gravité, pour notre système d'irrigation. Or ce n'est pas en un tel sens que nous avons entendu *effectivité* au cours de cette réflexion. Notre hypothèse (3) promet au contraire que c'est à partir de l'effectivité de la connaissance que doit être comprise l'effectivité des machines. Ne pas établir une telle déduction pourrait laisser penser qu'on a introduit un *deus ex machina* dans notre réflexion et, par un tour de passe-passe, supposé ce qu'il fallait démontrer. Il nous faut donc comprendre quel est ce *surcroît d'effectivité* qu'apporte la programmation, en nous assurant qu'il relève bien d'une possibilité interne de la connaissance, et non d'un principe externe qui serait subrepticement ajouté. Nous consacrons la section 13.3 à ce travail.

Il nous reste enfin à éclairer en quel sens toutes ces conclusions pourraient contribuer à certaines problématiques philosophiques contemporaines[27] que nous avons identifiées comme traversées par la même dialectique de la dépendance ou de l'autonomie entre un pôle « théorique » ou « intellectuel » de la connaissance, et un pôle « pratique », « incarné », ou « étendu ». Il ne s'agit bien sûr pas ici d'engager ces débats. Nous voulons seulement suggérer que la contribution positive de notre réflexion pourrait résider dans l'attention portée à articuler la continuité *et* la rupture qui existent entre ces deux pôles de la connaissance. Un seul exemple, celui de l'anthropologie cognitive, suffira à illustrer notre propos. Nous avons mentionné en introduction la thèse de la *projection d'organes*, introduite par le

27. Éclairage relatif aux pistes § 15, § 16, § 17, § 18, § 114.

philosophe allemand Kapp et reprise par Leroi-Gourhan[28] : les outils des premiers humains (et plus généralement des animaux évolués) sont des extensions et des remplacements de certains de leurs organes pour certaines de leurs fonctions (les dents pour déchirer, les ongles pour gratter, etc.). Cette thèse organise la grande fresque historique et conceptuelle qu'est *Le geste et la parole*, qui se termine sur l'idée que l'ordinateur en est l'étape ultime, prenant en charge la mémorisation et la transmission de ses « chaînes opératoires » (ses gestes techniques)[29]. Cette idée eut une grande influence sur la philosophie française de la technique, inspirant Derrida et Stiegler et, plus récemment, Bachimont, Steiner et Lassègue. Il ne s'agit pas ici de nier l'attractivité et la profondeur de cette thèse ; nous avons nous-même insisté sur l'émergence continue des techniques à partir du terreau des pratiques humaines. Cependant, à insister tant sur la *continuité* dans la longue durée de ce grand mouvement d'extériorisation des facultés humaines, on risque également de perdre de vue la *singularité* de la rupture épistémique que constitue l'approche systématique de la résolution de problème, qui fait violence aux écologies cognitives humaines, et amène à reconfigurer en profondeur leurs techniques et leurs institutions. Il ne s'agit plus ici de projeter, par un mouvement centrifuge, les fonctions essentielles de la personne humaine dans un environnement modelé pour son usage. Ce sont plutôt, avons-nous argumenté, les chaînes opératoires chères à Leroi-Gourhan qui prennent leur autonomie dès la révolution industrielle, et qui relèguent la personne à leur périphérie[30].

Il nous semble que cette articulation délicate que nous essayons ainsi de mettre en avant, entre continuité et rupture dans les pratiques humaines de connaissance, serait également intéressante à développer dans les autres problématiques mentionnées, par exemple concernant la relation entre sciences et techniques, ou entre connaissance et formes de vie.

Enfin[31], nous avons mentionné l'utilité possible de notre réflexion aux débats éthiques et politiques concernant l'empreinte du numérique dans notre monde humain ; la section 13.4 est visé à esquisser ces contributions possibles.

28. Voir plus haut, § 17.
29. (Leroi-Gourhan 1965, vol. 2, p. 53-56).
30. Voir plus haut, § 231.
31. Réponse à la question § 20.

13.2 L'apprentissage machine

Au début de notre réflexion[32], nous avions identifié quatre tendances de transformation de la programmation : la programmation simplifiée (par exemple visuelle), la programmation spécialisée, la programmation sur de nouveaux modèles d'ordinateurs, et l'apprentissage machine. Les trois premières tendances ont trouvé leur explication dans notre cadre général d'interprétation. Les deux premières doivent être comprises comme procédant de la possibilité interne qu'a la programmation de s'écrire dans tous les systèmes de signification des domaines qui la sollicitent, et de s'invisibiliser elle-même ainsi que la machine[33]. La possibilité d'ordinateurs qui ne soient pas construits sur des mécanismes de transmission a aussi été élucidée[34].

L'apprentissage machine est différent. Il remet en cause de manière assez profonde le principe même de la programmation – la recherche de solutions exactes et exhaustives d'un problème – et ainsi il est en général assez peu souvent qualifié de tel. Pourtant, comme nous l'avions remarqué en réponse à la thèse avancée récemment par Welsh[35] (« la fin de la programmation ») il convient nominalement aux définitions que nous avons données de la programmation comme « diriger les ordinateurs à faire ce qu'on veut », ou encore « agencer des signes afin de déterminer le comportement d'une machine[36] ». Qu'en est-il donc vraiment ?

Dans cette section, nous esquissons quelques pistes de réflexion sur ces méthodes, qui prennent aujourd'hui une place importante dans le débat public aussi bien qu'en philosophie, en prenant pour fil conducteur la question de leur similitude et de leur différence avec la programmation. Cela nous amène à distinguer l'apprentissage machine qui vise à résoudre des problèmes posés dans un cadre systématique, notamment scientifique (nous prenons l'exemple du problème de la structure géométrique des protéines) (§ 239) et celui qui traite de problèmes ouverts, comme la reconnaissance d'images ou la conversation (§ 240). Cette approche nous permet de dégager progressivement l'originalité profonde de l'apprentissage machine statistique comme forme de programmation. Sur la base de ces analyses, nous avançons la thèse de sa complémentarité essentielle avec la pro-

32. Voir plus haut, § 36.
33. Voir plus haut, § 196 et § 226.
34. Voir plus haut, § 214.
35. Voir plus haut, § 25.
36. Voir plus haut, § 39.

grammation classique pour la résolution de problème (§ 241).

§ 239. L'usage scientifique de l'apprentissage machine

Le principe de l'apprentissage machine statistique est très ancien, et dans une certaine mesure pré-date l'invention de l'ordinateur puisqu'on peut le rapporter à la cybernétique et aux neurones logiques de McCulloch et Pitts[37]. Nous parlons d'apprentissage machine *statistique* pour le distinguer de l'apprentissage machine dit *symbolique*, où une procédure de second ordre est explicitement programmée pour modifier le programme principal : tel était, par exemple, le principe du premier programme jouant aux dames, réalisé par Arthur Samuel[38], qui popularisa l'expression d'*apprentissage machine* : chaque partie était analysée par un second programme qui modifiait en conséquence les coefficients d'attractivité des situations ayant conduit à la victoire ou à la défaite. Une telle méthode correspond à ce que nous avons appelé *heuristique*[39] et ne remet nullement en cause notre épistémologie de la programmation.

Par contraste, les méthodes d'apprentissage machine statistique ont en commun de « programmer » un système de règles résolvant des problèmes à partir de seules données fournies par leur environnement. Dans certains cas, dits d'*apprentissage supervisé*, il s'agit de couples *entrées - sorties* qui donnent à la machine une idée des bonnes solutions pour des paramètres particuliers du problème, et lui demandent de dégager une fonction générale s'en approchant le mieux. Par exemple, on peut fournir à la machine des images en indiquant les animaux qu'elles représentent, le problème étant pour elle d'apprendre à reconnaître toute image animalière similaire. Dans d'autres cas, dits d'*apprentissage non supervisé*, il s'agit de laisser la machine organiser les données selon des fonctions prédéterminées – par exemple regrouper les éléments d'un jeu de données en sous-ensembles selon la proximité de leurs descriptions, ou au contraire trouver des corrélations ou des associations entre différentes dimensions descriptives des éléments. Par exemple on donne à la machine la liste des achats réalisés par des consommateurs sur un site de commerce en ligne, le problème étant pour elle d'établir une recommandation pour tout consommateur en inspectant son profil de consommation.

37. Voir à ce sujet la généalogie proposée par le livre référence de l'apprentissage profond*, (Goodfellow, Bengio et Courville 2016, p. 12-18).
38. (Samuel 1959).
39. Voir plus haut, § 199.

Il existe encore d'autres approches et d'autres familles de problèmes auxquelles on applique l'apprentissage machine statistique. Les techniques algorithmiques sont également nombreuses. En ce moment les réseaux de neurones* sont les algorithmes les plus connus et les plus polyvalents, en particulier les transformeurs* sur lesquels sont basés les grands modèles de langage* comme ChatGPT, mais d'autres méthodes sont également en usage, comme les algorithmes bayésiens*, les algorithmes génétiques*, ou les forêts aléatoires*. C'est de tout cet ensemble technique, extrêmement dynamique et foisonnant, que nous traitons ici.

Nous venons de décrire l'apprentissage machine statistique comme *programmation de systèmes de règles* résolvant des problèmes. C'est bien de cela qu'il s'agit si on considère leurs résultats, qui sont bien des programmes exécutés par des machines, et qui rendent des règles effectives au sein de situations investigatrices ou objectives – dans le sens dégagé par notre dernière partie. En diagnostic médical par exemple, on utilise des réseaux de neurones qui réalisent des inférences concernant certains aspects de cellules pour détecter la présence d'un cancer. Dans une usine, on les utilise pour planifier la maintenance des machines, en observant l'historique des pannes et de leurs causes. Dans le système judiciaire, ils peuvent – sujet très controversé – émettre des recommandations quant à des décisions de remise en liberté provisoire[40].

De ce fait, l'ensemble de nos analyses concernant les raisons de l'effectivité des ordinateurs au sein d'une situation continuent de s'appliquer. Peut-on en dire autant de l'interprétation contractuelle de la programmation ? Il semble que oui : le programmeur (ou plutôt, comme on l'appelle à présent, l'*ingénieur des données* (*data engineer*)) négocie une spécification avec le maître d'ouvrage qui doit permettre à celui-ci, certaines conditions concernant la situation étant satisfaites, de résoudre son problème.

Les choses sont en fait un peu plus compliquées que cela. Nous nous restreignons dans ce qui suit aux situations d'investigation scientifique, dont les différences avec la programmation sont plus simples à analyser. Nous abordons le cas général au développement suivant.

Une première différence apparaît entre programmation et apprentissage machine statistique dans le fait que le programmeur n'est plus en mesure de donner au maître d'ouvrage une garantie *logique* que son programme fonctionnera, mais seulement une garantie *statistique*. Ainsi, le programme AlphaFold, qui détermine

40. Sur ce sujet, voir (Pégny à paraître).

les structures géométriques des protéines à partir de leur formule chimique, associe un indice de confiance à ses prédictions, en fonction de la similarité du cas traité avec sa base de données[41]

Cette différence quant à la validité de l'inférence réalisée n'exige pas, à notre avis, d'ajustements conceptuels majeurs. De nombreux programmes développés de manière classique peuvent également résulter en inférences seulement probables, par exemple lorsqu'ils utilisent les méthodes statistiques de Monte Carlo* pour prédire l'évolution d'une situation. Une règle seulement probable, outre qu'elle est courante dans les affaires humaines, est tout à fait compatible avec l'idée d'une *règle contrôlée* et validée par un système scientifique. Il suffit en effet d'intégrer cette probabilité dans les raisonnements et les procédures qui l'utilisent ; les solutions résultantes devront être affectées d'un tel coefficient – cela ne les rendrait pas moins exactes et exhaustives. Dans la perspective de Dewey, on peut sans doute dire que la probabilité décrit l'inadéquation partielle du système de significations développé à rendre compte de l'expérience – mais cela ne remet pas en cause sa systématicité.

La différence importante se situe donc ailleurs – dans la *manière* dont le programmeur va modifier la machine afin qu'elle se comporte selon la spécification. Les algorithmes d'apprentissage machine statistique, essentiellement, tiennent compte *de l'ensemble* des données d'entrée pour se calibrer, et leur donnent des poids relatifs interdépendants – par exemple le mot

41. (Varadi et al. 2022). Il peut être ici utile de présenter cet exemple que nous allons utiliser de manière suivie dans ce développement. La détermination de la structure géométrique d'une protéine est un problème majeur de biologie organique, qui occupe des centaines d'équipes de recherche à travers le monde, et qui est souvent crucial dans la conception de médicaments. La fonction et le comportement d'une protéine peut en effet souvent être inféré de sa forme spatiale. Le problème est de prédire cette forme à partir de la chaîne d'acides aminés dont est composée la protéine (structure primaire). De nombreuses conformations de cette chaîne sont possibles du fait de la rotation libre de la chaîne autour de ses atomes de carbone. On distingue ainsi des structures secondaires apparaissant localement (hélices, feuillets), des structures tertiaires où la séquence se replie sur elle-même du fait d'attractions chimiques entre éléments lointains de la chaîne, et des structures quaternaires composées de plusieurs chaînes entremêlées. La détermination expérimentale de ces structures est difficile et coûteuse. Aussi le problème est-il devenu central en biologie computationnelle. La performance des différents modèles numériques est évaluée lors d'une conférence bi-annuelle CASP. Voir l'article Wikipedia. En 2020, le programme AlphaFold, présenté par la société DeepMind, a fait sensation en présentant un modèle fondé sur les transformeurs*, d'une performance bien supérieure à celle de tous ses rivaux.

« vol » aura une certaine valeur près de *« diamant »*, et une autre près de *« oiseau »*. Nous avons vu un ersatz d'une telle interdépendance dans le programme Deep Blue, qui tentait de programmer les interdépendances entre ces règles explicitement[42]. Les algorithmes d'apprentissage machine statistique renversent la démarche. Ils partent d'une présupposition d'*interdépendance entre toutes les données*, dont il s'agit alors de calibrer l'importance relative. Ce faisant, *ils ne relèvent pas de la science des opérations*, telle que nous l'avons définie[43], et notamment du principe de localité, qui exige qu'une règle soit définie seulement relativement *à quelques marques* de la situation, et non à son ensemble. Ces algorithmes, une fois calibrés, *sont une seule règle géante*, qu'on ne saurait décomposer en règles plus élémentaires.

On pourrait protester que, ces algorithmes étant des programmes écrits dans des langages de programmation classiques, ils obéissent bien aux principes que nous avons dégagés. Cependant ce n'est pas à cette *strate* de programmation que se situe l'apprentissage. Celui-ci n'enrichit pas ce programme de nouvelles instructions, mais modifie seulement des *coefficients* de la fonction représentée pour qu'elle s'approche au plus près de toutes des valeurs qu'elle doit simuler, comme le ferait une moyenne. L'apprentissage se situe ainsi à une strate intermédiaire entre le système de significations cible, par exemple la structure des protéines, et le langage de programmation. Il y a ainsi *une deuxième spécification* qui vient s'ajouter à celle de l'inférence qu'il faut représenter, celle qui concerne le processus même d'apprentissage : quantité de données disponibles, structure, fiabilité, etc. Il y a ainsi une *double* négociation avec le maître d'ouvrage, celle qui concerne de manière classique la fonction (ou l'inférence) proprement dite à réaliser, et celle qui concerne le processus d'apprentissage lui-même. De son côté le programmeur a également une double tâche, d'une part de trouver ou de construire l'algorithme adéquat qui permettra une induction satisfaisante, et d'autre part de réaliser le processus d'apprentissage lui-même. Il est entendu que toutes ces activités sont étroitement entremêlées – les méthodes agiles* sont d'ailleurs privilégiées dans ces processus de développement encore peu normés et industrialisés[44].

La seconde différence importante entre la programmation et l'apprentissage machine statistique porte donc sur la nature de la spécification ; en programma-

42. Voir plus haut, § 71.
43. Voir plus haut, § 162.
44. (Amershi et al. 2019).

tion classique, elle est constituée de significations censées décrire la situation problématique. En apprentissage machine statistique cela est également présent, mais ne sert qu'à la validation du programme ; la partie essentielle de la spécification est le jeu de données (le plus souvent une collection d'exemples résolus).

Il y a ici une rupture épistémique violente, car il s'agit à présent, si on transpose cette situation au plan de l'investigation humaine, de résoudre des situations problématiques *sans l'appui des significations*, comme des exercices répétés, dans des situations adéquatement variées, forgent une habitude unique capable de les résoudre toutes – comme on apprend à lutter en se confrontant à différents lutteurs, ayant chacun une morphologie et des tactiques différentes. S'il y a ici des règles, elles sont induites *a posteriori*, mais elles ne jouent aucun rôle dans la formation de la solution elle-même.

Cette analogie entre l'apprentissage machine statistique et l'induction empirique fut très tôt aperçue. Le connexionnisme (comme on appelait alors le projet d'intelligence artificielle fondé sur les réseaux de neurones) fut très tôt enrôlé par les opposants à l'hypothèse computationnelle de l'esprit comme preuve qu'un comportement intelligent pouvait se passer d'une formulation procédurale. Les partisans du renouveau des sciences cognitives y virent une possibilité de penser le caractère incarné et situé de l'intelligence, dans la lignée de Merleau-Ponty[45]. Sans entrer dans le débat de la pertinence de cette métaphore, nous notons seulement qu'une telle interprétation épistémologique de l'apprentissage machine statistique semble marquer le grand retour de l'associationnisme et du behaviorisme en psychologie, que les sciences cognitives avaient si fortement combattues à leur naissance, en argumentant de l'incapacité qu'aurait tout dispositif de renforcement de donner lieu aux structures de *contrôle* requises par l'intelligence.

La dualité de la spécification explique pourquoi l'apprentissage machine statistique peut bien être systématique (au niveau de la strate « classique » de programmation) tout en violant le principe de localité (au niveau de la strate d'apprentissage) – qui est le niveau dont dépend le résultat final. L'apprentissage machine statistique, selon la manière dont on le regarde, peut tour à tour apparaître comme *étant* ou *n'étant pas* de la programmation.

La violation du principe de localité entraîne une difficulté. Si les règles auxquelles on aboutit ne peuvent être décomposées en règles élémentaires, cela veut dire qu'elles ne se réfèrent pas à une théorie sous-jacente – et donc elles ne sont

45. (Varela, E. Thompson et Rosch [1991] 2016, p. 85).

pas systématiques en ce sens. On pourrait même avancer qu'elles sont chacune leur propre système, un seul immense modèle donnant d'un seul coup toutes les solutions de tous les cas particuliers. On peut penser ici aux modèles anciens que nous avions critiqués – le modèle arborescent et le modèle combinatoire, à leur ambition totalisante, corollaire de leur incapacité à être *génériques*, c'est-à-dire à *ne pas* donner la solution par avance à tout problème, mais seulement à créer un espace ouvert pour leur formulation et leur résolution. *De facto*, un modèle d'apprentissage machine statistique doit être entièrement recalibré si on souhaite qu'il intègre de nouvelles fonctionnalités.

Si tel est le cas, comment admettre l'usage de tels modèles dans la démarche scientifique ? Le débat épistémologique est en cours, et il n'est pas temps ici d'y participer. Nous avons déjà cité le constat intrigué de Cardon, Cointet et Mazières (2018) sur cette *technique sans théorie* que seraient les réseaux de neurones. Peut-on dire qu'on a une connaissance scientifique de lois obtenues par des convergences de séries statistiques, constituées de milliers de paramètres inutiles à toute explication structurelle ou causale ? Nous avons évoqué à ce propos la piste proposée par Napoletani, Panza et Struppa (2011) d'une *science agnostique* qui séparerait la résolution positive de problèmes de la compréhension structurelle des phénomènes. Nous ajoutons seulement la remarque suivante, qui peut sembler tautologique : les contributions de l'apprentissage machine statistique à la science, dont il est question ici, se déroulent tout de même dans un cadre scientifique. Par là nous voulons dire que, si les règles induites par la machine ne sont pas descriptibles elles-mêmes dans un système scientifique, leurs objets le sont de manière éminente – par exemple la composition chimique et la structure géométrique des protéines dans le cas d'AlphaFold. Ces règles héritent donc elles-mêmes, dans ces cas, de la structure elle-même fortement systématique de leurs objets (le langage des aminoacides et la stéréométrie). Réaliser des inférences par analogie de structures peut donc être une démarche empirique assez robuste ici, bien plus, par exemple, que lorsqu'on demande à ces algorithmes de réaliser des analogies sur des significations peu structurées, comme des images ou des phrases du langage naturel (nous y revenons).

Ainsi, le problème du repliement des protéines est l'objet de méthodes statistiques depuis plus d'une cinquantaine d'années[46] ; AlphaFold n'est donc une révolution dans ce domaine que par le bond impressionnant d'efficacité qu'il a permis.

46. Voir plus haut, la note 41 et ses références.

Cela suggère que, dans le cas de situations extrêmement « enchevêtrées », comme celles des jeux d'échecs et de go, ou comme celle justement des protéines, ces méthodes pourraient bien être ce qui se rapproche le mieux d'une loi empirique décrivant une relation régulière entre deux structures, *parce qu'*elles partent justement d'un modèle générique d'enchevêtrement qu'elles cherchent à adapter, plutôt que de règles élémentaires qu'il s'agirait de composer. Dans le cas des protéines, cela se voit par le fait que de nombreuses protéines *n'ont pas* de structure stable du fait de leur configuration chimique, et qu'AlphaFold, à juste titre, attribue un niveau de confiance très faible à ses propres tentatives de les représenter[47]. Ces algorithmes seraient de meilleures représentations des règles régissant ces structures enchevêtrées que des fonctions classiques car c'est eux qui encoderaient de la manière la plus compacte l'information déjà disponible à propos de certaines d'entre elles et sauraient le mieux l'extrapoler aux autres. On pourrait donc ne pas seulement les voir comme des méthodes inductives permettant d'obtenir des résultats intéressants, mais comme des dispositifs de représentation à part entière, au même titre que les scientifiques ont appris, progressivement, à considérer les nombres, les fonctions, les équations différentielles, les graphes, et les algorithmes comme des représentations adéquates de phénomènes naturels.

§ 240. Le cas des situations non systématiques

Dans les domaines systématiques où est utilisé l'apprentissage machine statistique, on peut encore parler de *contrat* passé entre le maître d'ouvrage et l'ingénieur des données. Le premier peut être limité quant au nombre et à la qualité de données qu'il peut fournir au second, mais ce défaut peut être évalué statistiquement, et le programmeur peut répondre au maître d'ouvrage par un engagement plus faible quant aux propriétés de la fonction recherchée et à son taux de confiance. Cela provient du fait que l'espace sous-jacent, celui des termes qu'il faut mettre en relation (dans notre exemple, les structures chimiques et géométriques des protéines) a bien une structure systématique, comme nous l'avons dit.

La situation est complètement différente dans les domaines non systématiques. Ici, les données d'entraînement font *toute* la spécification. En apprentissage supervisé notamment, le programmeur s'engage à ce que sa prédiction soit fiable sur ces données *et toutes données similaires*. Or cette dernière expression fait à la fois l'effectivité de l'apprentissage machine statistique et son problème, car une telle

47. (Terwilliger et al. 2023, p. 2), (Callaway 2022, p. 236).

spécification est intrinsèquement floue. On peut par exemple garantir que la machine reconnaîtra un animal dans 98% des images similaires à celles qui lui furent présentées lors de son apprentissage. Dans une telle garantie se trouve l'attente que la machine *aille au-delà* des images d'apprentissage – sinon le programme n'aurait aucun intérêt. Cependant si des images lui sont présentés dans des conditions *très* différentes (par exemple dans des situations de vie sauvage, tandis que la machine n'avait vu que des animaux en captivité) alors ce taux peut chuter de manière drastique. Ce que signifie *similaire* dans la spécification n'est pas spécifié, contrairement au cas du problème des protéines, dont l'espace est strictement délimité. Ce que sont des images *similaires* s'ancre au contraire dans un certain horizon d'attente humain, ouvert à l'ambiguïté et à l'indétermination, et c'est là-dessus que l'apprentissage machine statistique récent – celui notamment des larges modèles de langage – présente les résultats les plus étonnants.

Avant d'aller plus loin, on pourrait objecter qu'il est inhérent à tout processus de spécification de rencontrer des ambiguïtés. N'avons-nous pas nous-même suffisamment insisté là-dessus, et n'avons-nous pas justement présenté la programmation comme une entreprise de *systématisation* de la situation ? De la même manière, n'est-il pas naturel qu'on s'aperçoive, au cours de processus d'apprentissage machine, que le jeu de données proposé initialement était incomplet, ou biaisé, et qu'on l'enrichisse ainsi progressivement ? Toute spécification, dans les méthodes agiles*, est censée co-évoluer avec l'écriture du programme.

Une première réponse possible est qu'aucune borne n'est ici assignée à l'enrichissement possible des données puisqu'il se déroule hors d'un système contrôlé de significations. Il n'y a pas de jeu de données dont on pourrait affirmer qu'il « quadrille » suffisamment l'espace des possibles, car un tel référentiel manque *a priori*. On retrouve ici, encore une fois, le problème du cadrage[48].

Une telle réponse est cependant insuffisante. D'abord, la caractéristique des modèles récents est qu'ils semblent capables non seulement de réaliser des interpolations sur les données disponibles mais également d'inférer à partir d'elles la structure sous-jacente de leur espace, *fût-elle inductive*. Cela se voit bien dans la compétence qu'ont les grands modèles de langage* de générer des phrases (correctes) arbitrairement longues et emboîtées. Cela tient à leur capacité à analyser les structures sous-jacentes des textes fournis, non seulement entre mots adjacents mais également entre mots lointains et surtout entre groupes de mots adjacents ou

48. Voir plus haut, § 9, § 84, § 162.

lointains. Mémoriser ces structures leur permet d'encoder, d'une manière implicite, les règles de composition syntaxique du langage[49]. Il y a ici un point important à comprendre : ces modèles sont à juste titre appelés *génératifs* car ils peuvent manipuler des systèmes générés de manière inductive – ils sont capables de *systématicité* dans le sens que nous avons dégagé[50].

On peut affiner la réponse en concédant que le langage est certes un système, tout comme la structure chimique des molécules. Mais ce qui est vraiment ouvert et qui ne peut être résumé par des règles, fussent-elles inductives, n'est pas la structure superficielle du langage, sa grammaire, mais les relations entre les significations elles-mêmes, c'est-à-dire les structures profondes des textes. Or c'est là également que les larges modèles de langage présentent des résultats étonnants. Leurs modèles statistiques ne distinguent pas *a priori* syntaxe et sémantique, puisque ce sont les relations entre mots eux-mêmes (et non entre catégories grammaticales de mots) qu'ils modélisent[51]. Aussi les a-t-on découverts capables, depuis les célèbres modèles GPT, de « comprendre » la structure profonde d'un texte sur plusieurs niveaux, c'est-à-dire non seulement ce que que nous appelons sa *signification*, mais également son contexte, ses figures (ironie, métaphore, etc.), son genre (argument, narration, etc.).

On pourrait donc parfaitement contrer cette réponse en affirmant, dans une veine structuraliste, qu'il existe bien des *schèmes* invariants de la signification, inductifs comme les règles de la grammaire, mais actifs à plusieurs niveaux d'enchâssement – allant des objets de la syntaxe jusqu'à ceux de la rhétorique, voire de l'anthropologie, et que les grands modèles de langage* encoderaient implicitement – en proportion des textes fournis et du nombre de leurs paramètres. Certains, comme Piantadosi, un neuro-scientifique, avancent ainsi que ces modèles sont des *théories scientifiques* du langage :

> Ils sont les seuls modèles existants qui parviennent à capturer la dynamique de base du langage humain. Cependant, du fait qu'ils sont des réseaux de neurones, leur état – au moins initial – est totalement différent des règles et des principes qui ont dominé les approches génératives du langage. Comme décrit plus haut, leurs paramètres en viennent à incarner une *théorie du langage*, *y compris des représentations de l'état latent* par le biais d'une phrase et d'un discours. La même logique de réglage des paramètres pour formaliser et comparer les théories se retrouve dans d'autres

49. Sur ce sujet, voir l'introduction de (Li 2022).
50. Voir plus haut, section 9.4.
51. (Piantadosi 2023).

> sciences, comme la modélisation des ouragans ou des pandémies : tout ensemble d'hypothèses génère une distribution de prédictions et les hypothèses sont ajustées pour obtenir les meilleures prédictions possibles[52]. *(Nous soulignons)*

Piantadosi va même plus loin et affirme que ces modèles ne sont pas seulement des théories du langage, mais également des théories *de l'esprit humain* :

> En particulier, nous pouvons considérer chaque modèle ou ensemble d'hypothèses de modélisation comme une hypothèse possible sur le fonctionnement de l'esprit. Le fait de vérifier dans quelle mesure un modèle correspond à un comportement humain constitue alors un test scientifique des hypothèses de ce modèle. […]
> Ces modèles semblent promettre leur intégration possible à ce que nous savons d'autres domaines, en particulier la science cognitive et les neurosciences. [53].

On retrouve là la vieille idée d'Herbert Simon, que les programmes d'intelligence artificielle seraient la forme exacte que devraient prendre les théories psychologiques de la connaissance[54]. Nous ne souhaitons pas ici entrer dans la critique de cette position, qui nous semble retrouver les présuppositions de l'hypothèse computationnelle de l'esprit. Nous faisons seulement deux remarques, en admettant que de tels modèles donnent des prédictions *fiables* de ce que les personnes humaines feraient ou diraient de manière régulière, dans tel ou tel contexte (ce qui est une admission immense).

D'abord, on ne pourrait parler de *théorie*, selon le sens qu'on attribue habituellement à ce terme, que s'il était possible d'extraire de ces modèles des schèmes inductifs sous-jacents, c'est-à-dire s'il était possible de décomposer la règle géante qu'ils représentent en règles élémentaires (locales) qui pourraient être composées entre elles. Dans le cas contraire, on devrait souscrire à l'idée d'une science sans théorie, ou d'une science agnostique[55].

Ensuite, quand bien même ces schèmes seraient exhibés, et même s'ils fournissaient une théorie adéquate de l'intelligence et du comportement humain *actuels*, il se trouve que ceux-ci évoluent *dans le temps* et que de nouveaux concepts et de nouvelles structures de significations *arrivent* – il suffit de prendre pour exemple les idées même de *machine* et de *système* qui n'arrivent à leur maturité que progressivement, comme nous avons tenté de le montrer. Un modèle entraîné seulement sur des textes du XVI^e^ siècle aurait sans doute des difficultés à dialoguer avec des

52. (ibid., p. 9).
53. (ibid., p. 11-12).
54. Voir plus haut, § 70.
55. (Napoletani, Panza et Struppa 2011).

personnes d'aujourd'hui, quand même on lui expliquerait tous les nouveaux mots et tous les nouveaux concepts. En d'autres termes, nous avançons la thèse que les structures de la pensée et du comportement humains sont développées *dans le temps historique*, en interaction avec les structures culturelles, sociales, économiques, etc. (ce qui ne veut pas dire qu'elles sont déterminées historiquement par celles-ci). Elles sont ouvertes à l'horizon du monde. Comme le dit Dewey, le langage n'est pas un système, mais une « constellation » de significations, et il échappe à toute systématisation parce qu'il est le sol sur lequel se déposent les strates successives de significations qui témoignent des interactions, sans cesse renouvelées, des personnes avec leur monde. C'est cette historialité qui rend toute mise en système de l'être humain intrinsèquement provisoire et faillible.

En d'autres termes encore, ce qui empêcherait les machines de penser et d'être « créatives » n'est pas que certaines de nos facultés ne peuvent être modélisées par des règles, mais qu'elles ne sont pas *situées* comme nous dans un monde qui leur pose des problèmes : c'est là à peu près, sur un plan différent, l'argument que John Searle opposait à l'intelligence artificielle il y a une quarantaine d'années[56]. C'est surtout l'argument qu'utilise Daniel Andler dans son ouvrage récent sur l'intelligence artificielle[57].

Par contre, si on venait à concevoir des machines persistantes sur la longue durée, dotées d'objectifs de très longue durée, alors il serait tout à fait possible qu'elles en viennent à développer des formes d'investigations qui leur soient propres, et qu'elles en viennent à être « créatives ». Mais cette créativité se situerait sur une ligne historique différente de la nôtre, qui serait fonction de la forme de leurs interactions situées avec le monde. L'*imitation* de l'intelligence humaine, ancrée dans la perception et le métabolisme du corps humain, leur serait un détour malcommode et un frein ; ce qu'on leur demanderait serait seulement de nous *comprendre*, mais pas de nous *imiter*, comme nous avons tenté de l'indiquer par notre expérience de pensée sur le véhicule autonome[58]. On peut se référer au beau film d'anticipation *Her*, de Spike Jonze (2014), pour suivre un développement possible de cet argument.

56. (Searle 1980, p. 422).
57. (Andler 2023), en particulier chapitres 7 et 8.
58. Voir plus haut, § 231.

§ 241. Complémentarité avec la programmation

Ces réflexions nous permettent de revenir à la question de la *fin de la programmation* annoncée par Welsh[59]. Nous pouvons partir de la remarque que fait D. Jackson dans un billet récent à ce sujet :

> GPT est très doué pour écrire du code qui ressemble à tout le code déjà écrit auparavant. Personne n'a montré d'exemple où [ce modèle] inventerait de nouvelles et de meilleures façons d'écrire du code, donc si nous abandonnons tout à GPT-4, notre code stagnera au niveau des connaissances que StackOverflow [note : *un site populaire d'entraide pour programmeurs*] avait en 2021, lorsque la formation de GPT-4 se termina. Et si nous cessons d'innover, une future version de GPT n'en saura pas plus[60].

Cette assertion se situe dans la droite ligne de notre développement précédent. Elle place d'emblée la programmation non pas dans la perspective fermée d'un problème particulier, comme peut l'être celle de la géométrie des protéines, mais dans la perspective ouverte de l'investigation elle-même. Pour Jackson, les grands modèles de langage* automatiseront l'écriture routinière de code, libérant le programmeur pour ce qui constitue le cœur de son activité, *i.e.* la spécification et l'architecture du logiciel. Or la spécification, dans notre interprétation contractuelle de la programmation, est bien le cœur de cette forme d'investigation visant à insérer une machine dans une situation problématique. Cette investigation est intrinsèquement ouverte car elle passe par une négociation concernant la situation elle-même et ce qu'on peut raisonnablement attendre d'elle, en termes de régularité. C'est cette investigation qui est fondatrice de la programmation et, tant que de nouveaux problèmes seront posés, on peut supposer que de nouvelles structures procédurales devront être dégagées, et donc également de nouvelles théories d'opérations, *i.e.* de nouveaux langages.

Une autre manière de voir cette complémentarité est de réfléchir à ce que serait un système de règles statistiques. On peut par exemple imaginer un réseau d'ordinateurs, chacun implémentant une fonction grâce à l'apprentissage machine statistique, et organisés dans un flux de tâches pour réaliser une fonction globale – par exemple, produire une commande. Si le nombre de leurs interactions est élevé, la fiabilité du système dépendra de la *composition* des incertitudes. Supposons par exemple que chaque fonction soit fiable à 99% et qu'elle *bloque* le flux de tâches

59. Voir plus haut, § 25.
60. (D. Jackson 2023).

dans les cas restants. Si le flux de tâches sollicite 10 fonctions, sa fiabilité globale sera seulement de 90%. Il se peut au contraire que les fonctions interagissent entre elles de manière *à compenser* leurs erreurs, et non à les propager. Dans ce cas, la composition du flux de tâches aura l'effet inverse, et augmentera la fiabilité du processus d'ensemble.

Cette remarque vise à montrer que, même dans le cas où la programmation de fonctions individuelles serait remplacée par de l'apprentissage machine statistique, des considérations architecturales d'ensemble resteraient nécessaires. Ici on pourrait objecter qu'il serait possible de s'en passer, en appliquant l'apprentissage machine statistique directement au problème résolu par le flux de tâches – d'entraîner directement une fonction géante à son propos. Seulement une telle approche suppose qu'on possède des données en nombre suffisant pour entraîner une machine à ce problème global – ou que le temps pour les générer soit inférieur à celui de l'investigation proprement dite qui *compose* la solution.

Il y a ici une opposition très élémentaire entre deux schémas de résolution de situations problématiques, que nous avons entrevue lors de notre lecture de Dewey. Il y a d'abord l'adaptation par essais et erreurs, le schéma élémentaire de la vie, qui *sélectionne* les solutions adéquates. Il est très important de noter que ce schéma n'exclut absolument pas la capacité de l'organisme à réaliser des inférences – à anticiper le développement de la situation. Ce qui est simplement exclu, c'est sa capacité à *contrôler* ces inférences, c'est-à-dire à les manipuler sur un plan symbolique où leurs conséquences, ainsi que leur éventuelle invalidité, peuvent être aperçues *avant* que l'action soit entreprise.

Un tel contrôle de l'inférence est le propre du second schéma de résolution, celui du raisonnement. L'apprentissage machine statistique est très habile à générer des inférences extrêmement fines à partir d'essais et d'erreurs, si bien qu'il dote la machine de structures similaires à ce que nous appelons *contrôle*, et cela à plusieurs niveaux de profondeur. Mais il n'y a là pas plus de contrôle qu'on ne dirait d'un animal capable de voir deux fois plus précisément qu'un autre, qu'il est capable de *critiquer* les représentations de ce dernier.

C'est sans doute là la raison qui explique la faible performance (actuelle) de l'apprentissage machine statistique dans la résolution symbolique de problèmes[61]. Ces problèmes ne sont pas *conçus* pour l'apprentissage par essais et erreurs, mais pour une résolution *exacte* qui passe par des procédures de *contrôle* des inférences.

61. (Arora, Singh et Mausam 2023).

Ainsi, un vieux problème scolaire utilisé par la première intelligence artificielle :

> Marie est deux fois plus âgée qu'Anne ne l'était lorsque Marie avait l'âge qu'Anne a aujourd'hui. Si Marie a 24 ans, quel est l'âge d'Anne[62] ?

tient ChatGPT en échec à plusieurs reprises[63]. De manière intéressante, le programme se comporte assez scolairement, *imitant* les méthodes de résolution de tels problèmes. On a ainsi l'impression d'avoir affaire à un écolier *qui sait* comment les bons élèves se comportent pour résoudre de tels problèmes, mais qui ne sait pas réfléchir au problème lui-même. Il les imite en espérant « donner le change ». Même après trois indications complémentaires, GPT 3.5 se montre incapable de trouver la solution.

Ces déficiences donnent une assez bonne impression de la forme *native* des inférences dont sont capables nativement ces algorithmes. Bien sûr, il est probable que d'ici quelques années, voire quelques mois, elles seront corrigées. Deux pistes sont (entre autres) examinées à ce sujet. La première, appelée *auto-affinement* (*self-refinement*), vise à laisser le robot reprendre ses réponses de manière itérative ; pour l'instant, une telle méthode n'est pas effective sur les raisonnements mathématiques[64]. La seconde piste vise à laisser le modèle déléguer certaines tâches à d'autres logiciels qu'il identifie comme pertinents pour résoudre la tâche identifiée, par exemple une calculatrice, un moteur de recherche ou un agenda. On apprend simplement à ce modèle quand et comment utiliser ces outils, par quelques exemples, dans la continuité de l'apprentissage machine statistique. Ici, le modèle fait des progrès importants, en particulier dans les tests mathématiques, simplement grâce à l'usage de la calculatrice[65].

Il est bien sûr trop tôt pour tirer des conséquences de recherches qui viennent d'être engagées. Cependant, selon notre raisonnement, c'est la seconde voie qui devrait être la plus féconde pour augmenter les performances de ces machines dans la résolution de problème, car elle met en place une *collaboration* entre les deux formes essentielles de l'investigation locale, l'une homéostatique, qui *converge* vers un état d'équilibre avec son milieu, et l'autre stratégique, qui le *planifie*. Ces deux approches sont complémentaires. La programmation est orientée vers la systématisation, c'est-à-dire vers le *contrôle* de la situation par la mise en évidence d'un

62. (Minsky 1967). La réponse est 18 ans.
63. En juillet 2023.
64. (Madaan et al. 2023).
65. (Schick et al. 2023, p. 6). Dans le tableau 4 de cet article, on voit que la performance multipliée entre $\times 2$ et $\times 5$ selon les modèles comparés et les tests.

noyau d'inférences valides et sûres à son propos, sur lesquelles le raisonnement peut se déployer. L'apprentissage machine statistique est au contraire tout entière occupée d'*adapter* l'algorithme à une succession indéfinie de cas, négociant chaque fois un compromis entre la somme des cas passés et le cas présent qui minimise sa fonction de coût (distance, temps, mémoire, etc.). Une telle approche, si elle ne garantit jamais le succès, permet cependant d'être ouverte à de nouvelles situations et de *tenter* d'y apporter une réponse, tandis qu'une approche systématique est mise en échec par l'imprévu.

Il ne nous semble pas ici déplacé de parler de *style* de résolution de problème. Alors que le fonctionnement de programmes traditionnels, tout entier tourné vers l'effectivité, peut devenir assez rapidement opaque pour ses interlocuteurs humains, et en retour les rendre *obtus* à tout changement de contexte, l'apprentissage machine récent résout des problèmes d'une manière familière au sens commun – ce qui lui donne un aspect plastique, sensible et humain. Dialoguer avec ChatGPT donne l'impression de parler à une personne dotée d'un savoir livresque immense, cherchant à plaire à son interlocuteur, et raisonnant toujours par association d'idées : agençant de manière adroite des bouts de réponses à des questions similaires, mais doué de peu d'esprit critique (de capacité à *contrôler* le sens de ce qu'il dit). À ce titre, il se pourrait bien que d'ici peu, la qualification d'*opacité* soit abandonnée à propos de ces modèles, étant donnée la bonne volonté avec laquelle ils explicitent leurs raisonnements et leur associations de concepts.

Il serait donc possible que l'idée avancée par D. Jackson d'une *collaboration* entre programmation et apprentissage machine statistique soit capable de très nombreuses configurations. Lui-même évoque un second rôle que peut jouer un grand modèle de langage*, non seulement celui de générer du code, mais également celui de détailler une spécification, s'il connaît assez bien le domaine en jeu (D. Jackson prend ici l'exemple de la spécification d'un réseau social, un concept bien maîtrisé et bien connu). Il n'est donc pas aisé d'anticiper de quelles manières s'organiseront dans le futur les résolutions de problèmes, et quelle sera la part respective laissée à la programmation et à l'apprentissage machine statistique.

Il est intéressant de prolonger cette idée de collaboration entre *styles* de résolution de problème par une analogie avec la distinction que font certains psychologues entre deux modalités de contrôle des processus cognitifs humains, appelés *système 1* et *système 2*, « l'un rapide, automatique et non conscient, l'autre

lent, contrôlé et conscient, qui opèrent de manière largement indépendante[66] ». L'apprentissage machine statistique et la programmation, pourrait-on arguer, se trouvent dans le même rapport : elles sont deux stratégies possibles dont nous disposons à présent pour apprendre aux machines à réaliser des inférences, l'une qui passe par l'entraînement d'une « habitude » à une certaine classe de situations problématiques, l'autre par leur description systématique et leur mise sous contrôle par un système de règles.

Une troisième image que nous soumettons à la lectrice est celle des deux *droïdes* de la célèbre série de films *la Guerre des étoiles*, C3PO et R2D2. Le premier, ayant une forme humanoïde, est très bavard. Il sait parler toutes les langues et a une mémoire culturelle et historique gigantesque. Il est spécialisé dans la médiation entre civilisations, et notamment quant à leurs protocoles d'interaction. Il est par contre incapable d'aligner deux inférences à la suite, et encore moins des calculs. Il analyse souvent très mal les problèmes, qu'il associe aux situations protocolaires dont il a l'expertise. R2D2, au contraire, ne peut prononcer un seul mot – seulement quelques *bips* de tonalité variable, et a besoin en général de son compère pour communiquer avec les humains. Il est par contre capable de résoudre tout type de problème mathématique, informatique ou mécanique. Le contraste cognitif fort qui caractérise ce duo comique de robots nous semble anticiper de manière heureuse la probable complémentarité des deux formes essentielles de programmation que nous avons identifiées.

13.3 L'effectivité des règles, des machines, des systèmes

Dans cette section, nous revenons sur la question du *sens* et de la *déduction* de l'effectivité des machines, à partir de celle de la connaissance[67]. Nous commençons par revenir sur la tension de la connaissance vers la systématicité, dont relèvent aussi bien la programmation que l'apprentissage machine statistique (§ 242). Nous l'analysons comme une recherche de cohérence de l'investigation, ou encore d'une *cohérence des intentions* (§ 243). Avec l'exécution automatique de ratiocinations, cette cohérence est en partie *gagée sur la machine*, puisqu'on doit supposer qu'elle est construite conformément à l'intention du programme (§ 244). Nous analysons enfin l'effectivité des machines comme une forme de *détour productif* dont

66. (J. Evans et Frankish 2009, p. v).
67. Voir plus haut, § 238.

le schéma caractérise l'ensemble de la connaissance pratique, celui de *construire* des moyens, symboliques ou matériels, pour résoudre des classes générales de problèmes (§ 245).

§ 242. L'émergence et l'expansion de la systématicité

On pourrait croire que l'apprentissage machine statistique, en entraînant les machines à imiter l'inférence associative humaine, avec ses forces et ses faiblesses, *tourne le dos* au projet de systématisation de l'expérience auquel nous avons associé la programmation. Il n'en est rien. Il y participe pleinement, en permettant l'extension du règne des inférences automatisées à des situations dont la description systématique semblait hors de portée. Quelle que soit la monstruosité (pour la capacité analytique humaine) des règles géantes proposées par l'apprentissage machine statistique, elles sont tout de même des procédures effectives traitant d'investigations que *nous* ne savons pourtant pas récapituler de manière systématique. Il y a là comme un détour que prennent les mathématiques pour résoudre des problèmes là où nulle solution ne semblait possible. Les statistiques sont le moyen de ce détour ; elles furent d'ailleurs elles-mêmes inventées par les mathématiciens du XVII^e^ siècle pour *contourner* le problème du hasard, qui semblait rendre impossible l'énoncé de règles fermes à propos d'un vaste champ de phénomènes naturels et sociaux[68].

Il y a là *à nouveau* une extension de la systématicité au-delà de ce qu'on pouvait penser être son domaine naturel. Nous avons vu d'abord comment la machine permettait d'appréhender systématiquement des phénomènes chaotiques qui semblaient lui être intrinsèquement irréductibles – qu'on appelait justement *systèmes complexes* pour signifier ce projet[69]. Au chapitre précédent, nous avons vu comment la machine (interactive) avait permis d'étendre une deuxième fois ce domaine hors du champ fonctionnel strict (procédures à *entrées - sorties*), qui ne permettait pas de décrire les règles et les procédures de la plupart des pratiques humaines. Il est ainsi devenu possible, contrairement à toute attente, de définir des procédures qui dépendent de l'évolution indéterminée de la situation, si seulement l'*espace* de sa variabilité (et non nécessairement sa *loi*) pouvait être décrit selon des règles inductives. Par là, ce sont les pratiques humaines, dont on voyait bien mieux l'extraordinaire adaptabilité aux circonstances que leur régularité sous-jacente, qui

68. (Hacking [1975] 2006).
69. Voir plus haut, § 216.

furent mises à portée de l'automatisation. Avec l'apprentissage machine statistique, une troisième expansion de la systématicité est réalisée, cette fois, vers les domaines de la sensibilité et de la sociabilité humaines, *bien que ceux-ci ne puissent être décrits dans des systèmes de règles*.

De multiples questions sont possibles concernant la force sous-jacente à cette expansion : D'où provient-elle ? Apparaît-elle dans le temps ou est-elle inhérente à l'esprit humain ? Ses conditions de développement sont-elles historiques ? Kant, dans une formule restée célèbre, l'avait interprétée comme une tension propre à l'entendement humain, « à tout moment affairé à scruter les phénomènes dans l'intention de déceler en eux quelque règle[70] ». Le terme allemand utilisé par Kant, *durchspähen*, évoque l'idée d'une scrutation *qui parcourt* l'horizon afin d'y déceler la moindre trace d'une régularité, et *jederzeit* (*à tout moment*), préféré au plus banal *immer* (*toujours*), évoque le qui-vive.

Nous n'entendons pas bien sûr traiter ces questions ici, ni entrer dans le riche débat des historiens des idées et de la culture concernant la notion très critiquée de *miracle européen*, sa singularité, et ses causes possibles[71]. Il n'est pas non plus question de discuter l'*interprétation* de ce fait, notamment celle de Heidegger qui l'associe étroitement au *projet* de la métaphysique. Nous cherchons seulement à savoir pourquoi l'idée d'une résolution *systématique* de problèmes, qui semble ne pas devoir dépendre du progrès technique matériel, apparaît néanmoins si tardivement dans notre histoire intellectuelle.

Notre réflexion permet de mettre en avant trois remarques utiles à cette discussion.

D'abord, Platon et Aristote formulent nettement cette idée systématique, quoique le second la limite aux investigations scientifiques. Ce n'est pas qu'ils n'ont pas l'idée de l'appliquer aux affaires humaines : Selon Martha Nussbaum, la technique (*τέχνη*) est conçue d'emblée par les Grecs comme l'ensemble des entreprises mobilisant la raison pour contrôler l'effectivité de l'action et la soustraire aux aléa de la fortune (*τύχη*)[72]. Cependant Aristote, plus que Platon, est sceptique quant à la systématisation possible des techniques, car elles sont pour lui des savoirs dotés d'une universalité *faible*. Sans même évoquer la

70. *« Dieser ist jederzeit geschäftig, die Erscheinungen in der Absicht durchzuspähen, um an ihnen irgendeine Regel aufzufinden »*. *Critique de la raison pure*, A 126, Ak. IV, 92, notre traduction.
71. Voir, à ce sujet, les discussions de l'ouvrage collectif de (Roberts, Schaffer et Dear 2007) ; une thèse récente est celle de (Mokyr 2016).
72. Voir plus haut, § 98.

métaphysique aristotélicienne de la contingence, l'observation de la diversité et du caractère historiquement situé des écologies cognitives humaines[73] donne à cette opinion une très forte plausibilité. Les difficultés qu'eurent les réductions en art à formuler des méthodes universelles de résolution de problèmes pour des domaines particuliers confirment le bien-fondé de cette position de bon sens.

Ici se trouve notre deuxième remarque, à savoir, tout simplement, que cette idée est *difficile*, et que les idées difficiles prennent du temps. Il suffit de penser au temps pris par les mathématiciens pour dégager l'idée de l'algèbre dans toute sa généralité. Or cette idée était requise afin que Lovelace puisse parvenir à l'idée qu'il est *sérieusement* possible de « géométriser » nos plans d'actions, et entrevoir la possibilité d'une *science des opérations*. Il était également requis que le regard porté sur les pratiques humaines soit transformé, et qu'on commence à imaginer de les rendre absolument et exhaustivement contrôlables en entreprenant de reconfigurer rationnellement, non seulement leurs méthodes, mais également les environnements et les situations dans lesquelles elles doivent s'appliquer.

Il n'est certainement pas indifférent, à ce sujet (et c'est notre troisième remarque), que cette révolution du regard pratique se fasse au même moment que celle du regard scientifique, déjà évoquée[74]. En particulier, nous restons dans l'opinion commune en affirmant que le contact décisif entre les sciences et les techniques eut lieu autour de l'idée de *machine*. C'est en ce lieu que la possibilité de la mathématisation des phénomènes naturels aussi bien que des artefacts est conçue dans toute sa clarté, ce qui nous ramène à l'idée que le *concept* de *machine* joue un rôle essentiel dans l'idée d'une systématisation possible des pratiques. Il ne serait donc pas exact de dire, avec Knuth, que l'informatique aurait dû exister avant l'invention des ordinateurs. Qu'on conçoive la possibilité universelle de la résolution systématique de problèmes ne dépend certes pas de l'existence *empirique* de machines capables d'exécuter toute procédure, mais qu'on en ait dégagé le concept dans toute sa clarté et dans toute son universalité[75].

§ 243. La cohérence des intentions

Les remarques que nous venons de faire permettent d'introduire la thèse que, quelles que soient les ruptures épistémiques qui peuvent exister entre délibération,

73. Voir plus haut, section 5.1.
74. Voir plus haut, section 7.5.
75. Voir également plus haut, notre remarque en fin de § 191.

ratiocination technique et calcul systématique, elles continuent de relever d'une même effectivité de la connaissance, dont elles transforment seulement la modalité ; et c'est en particulier le cas de l'effectivité des machines, modalité caractéristique de la résolution systématique des problèmes.

Cette thèse peut également être exprimée comme suit : ce qui fait l'effectivité d'une machine n'est pas la puissance de sa source d'énergie, la solidité de ses matériaux, la sophistication de ses technologies, ou encore la qualité de son opération ; tous ces principes ne deviennent effectifs que parce qu'ils sont intégrés *de manière cohérente* à l'intention de la machine.

Cette *cohérence des intentions* est à l'œuvre aussi bien dans le calcul systématique que dans la ratiocination technique et la délibération.

En effet, lorsqu'une erreur de calcul est réalisée, on cherche quelle règle a été mal appliquée (en quoi on n'a pas été cohérent avec le système de règles qu'on s'était donné), ou alors on regarde quel chiffre a été mal lu ou recopié (en quoi on a réalisé une erreur d'« interface » entre le système de règles et la situation sensible à laquelle on l'appliquait). De la même manière, quand une ratiocination technique échoue, on vérifie si la procédure a été bien respectée, ou on inspecte la situation pour voir si celle-ci se trouvait dans une conformation que la procédure n'avait pas prévue. Enfin, lorsqu'une délibération débouche sur un résultat contraire, on s'assure d'abord qu'on n'est pas victime d'une erreur de raisonnement (comme ne pas avoir perçu les deux sens possibles d'un même mot), ou alors on incrimine un fait qu'on ne connaissait pas, ou qu'on avait mal interprété ; c'est-à-dire, encore une fois, non pas le raisonnement lui-même, mais son inadéquation à la situation à laquelle on l'avait appliqué.

Il y a là un *même schéma* qui se répète aux trois niveaux de l'investigation que nous avons dégagés. Il est d'une part possible de se contredire, notamment lorsqu'on n'a pas procédé conformément aux injonctions de significations exactes, c'est-à-dire préalablement vérifiées dans un système. D'autre part, les erreurs de jugement peuvent relever d'une *mauvaise lecture de la situation*, c'est-à-dire du fait d'*appliquer* à celle-ci un raisonnement, une procédure, un système de règles qui ne lui conviennent pas, autrement dit qui n'aboutissent pas à la situation qu'on espérait obtenir par cela. Il peut s'agir, non seulement d'une erreur *locale* d'attribution, comme dans les exemples ci-dessus, mais également d'une erreur *globale*. Quand j'applique par exemple les règles du calcul décimal à des nombre hexadécimaux, je ne parviens pas au résultat escompté, non parce que les règles sont fausses, mais

parce qu'elles ne sont pas adaptées à la situation problématique que j'essaie de résoudre.

On voit ici que ce qui change dans le passage de la délibération à la ratiocination technique, et de celle-ci au calcul systématique, c'est l'*ampleur* du compact de significations attribué à la situation, et donc également celle du *pari* qui est réalisé. À l'ensemble de *faits* que la délibération juge valides afin d'en faire la matière de son raisonnement, la procédure technique ajoute une séquence d'opérations décrivant le comportement attendu de la situation ; le système de règles, quant à lui, projette sur la situation tout un « modèle » ou toute une « théorie » quant à son comportement. À ce différentiel d'*amplitude* sont directement corrélés ceux d'*effectivité*, mais également de *risque* et de *difficulté* qui sont communément constatés entre jugement délibératif, technique, et systématique.

Tout ce que nous disons là est bien connu depuis les débuts de la logique. D'un syllogisme simple à une chaîne de raisonnement, et de cette chaîne à un système assis sur des premiers principes, c'est chaque fois l'ampleur des conséquences possibles, mais également la difficulté et le risque qui croissent avec le nombre de prémisses à pré-valider avant qu'on puisse les attribuer à l'expérience. Mais il est également une autre manière de voir cet accroissement d'amplitude, qui passe par le *Pont aux ânes*[76]. Cette « machinerie logique » affirme en effet que la procédure de constitution du syllogisme réussira à coup sûr *si les listes des termes décrivant le problème sont elles-mêmes bien constituées*. On pourrait objecter à Aristote qu'on n'a rien gagné par là. Cependant cela serait faux : car le problème de bien raisonner a été transformé en celui de bien décrire. On a formé un petit noyau de certitude au sein de la situation en échange d'exigences différentes à son égard. Ce mouvement une fois initié, rien n'empêche de l'étendre progressivement, par composition, et d'augmenter ainsi peu à peu la part que prennent les cellules de certitude dans le milieu problématique, de manière à former des *chaînes* et bientôt des *réseaux* de raisonnements possibles. Par ce mouvement, l'instabilité d'une classe de problèmes donnée est peu à peu circonscrite et décrite par des conditions qui vont être factorisées ensemble et *extraites* des problèmes eux-mêmes, par leur localisation dans ce qui va être appelé la *situation*. Ces conditions vont constituer un espace de problème qu'une *machine inférentielle* peut parcourir librement sans se soucier de l'accord avec l'expérience, car elle considère ces conditions comme de simples paramètres. Son critère de vérité est seulement qu'elle applique correc-

76. Voir plus haut, § 104.

tement les règles qui la définissent, ce qui n'est rien d'autre que la simple cohérence avec soi-même, avec ses propres intentions.

Il y a ainsi une division du travail intellectuel. Il y a d'abord le travail, qui vaut une fois pour toutes sur une classe de problème donnée, de préparer des systèmes de règles où des procédures exactes de raisonnement peuvent être développées et examinées. Ce travail étant réalisé, il reste au jugement, dans chaque situation, à transcrire les termes de son problème dans ceux du système considéré, puis à effectuer une ratiocination suivant les prescriptions de la solution. Cette idée correspond à la conception des notations qui émerge chez les algébristes anglais au début du XIX^e siècle, c'est-à-dire à cette idée que les raisonnements peuvent progresser par la manipulation selon des règles de symboles non interprétés. Selon Michael Detlefsen, cette conception « instrumentaliste » des notations prépare l'idée hilbertienne des systèmes de calcul formel[77].

Cette division du travail permet d'expliquer pourquoi on appelle les procédures systématiques *effectives* et quel est le lien entre le sens informatique de cet adjectif et le sens plus général dans lequel nous l'avons pris dans ce travail[78]. Qu'on caractérise comme telles les procédures construites selon les règles d'un système ne signifie pas que toute autre procédure ne le serait pas. Toutes nos connaissances, en tant que règles, ont un certain degré d'effectivité, ainsi que nous l'avons rappelé plus haut[79]. Simplement, cette caractéristique signifie que le système leur accorde une *garantie d'effectivité*, grâce au travail préliminaire qui a été accompli lors de sa construction. Cette garantie n'est rien d'autre que celle attendue de la division même du travail entre investigations globales, qui vérifient les règles et les assemblent en système, et les investigations locales qui les mobilisent dans un tel cadre. Bien entendu, une telle garantie ne vaut que dans la mesure du crédit qu'on apporte à son émetteur. C'est pour cela qu'il est possible de douter des déductions des systèmes d'astrologie, aussi cohérents et élaborés soient-ils.

L'effectivité des machines est de cette même nature. Les machines « en soi » ne sont pas plus ni moins effectives que n'importe quel autre phénomène naturel dans le monde ; ce qui est effectif *en elles* sont les intentions que nous gageons *sur elles* ; et elles ne nous donnent donc jamais qu'une *garantie d'effectivité*. Ce sont ces assertions que nous allons à présent tenter d'expliciter.

77. (Detlefsen 2007, p. 263-277).
78. Réponse à la question § 3.
79. Voir plus haut, § 238.

§ 244. L'effectivité des machines et la cohérence des intentions

La conception instrumentaliste des notations a encouragé l'assimilation de la programmation à une activité syntaxique ou formelle, qui ne se préoccupe pas de sémantique et de significations[80]. Tout notre travail tend à montrer, après d'autres, combien cette vision est partielle. La phase de *codage* peut être en effet dite formelle parce que les significations de la situation sont déjà préparées pour constituer un espace de problème pour la machine, qu'il n'y a plus alors qu'à diriger dans cet espace selon les règles qu'elle autorise[81]. Cependant, le codage reflète seulement le partage de responsabilité qui résulte d'un accord négocié entre maître d'ouvrage et programmeur. Cette phase de spécification, qui précède le codage, ainsi que les phases d'implémentation, de test et de production, qui le suivent, sont éminemment « sémantiques ».

Dans l'interprétation contractuelle de la programmation, c'est en effet la *négociation* entre le programmeur et le maître d'ouvrage qui fixe la frontière qui sépare la machine de la situation. Par là sont séparés un *intérieur* et un *extérieur*, le premier étant un lieu régi par des règles certaines et engageantes, le second seulement par des hypothèses intentionnelles. La spécification permet au maître d'ouvrage de dire : *« Si la situation (par exemple l'utilisateur) se comporte ainsi relativement à la machine, alors celle-ci fera telle ou telle chose qui modifiera la situation en conséquence »*.

Pour le maître d'ouvrage, la machine a donc le même rôle, *dans l'expérience*, que la ratiocination (humaine) dans l'investigation. La machine est pour le maître d'ouvrage une *garantie de solution* et représente un espace régi par des règles d'une certitude *supérieure* à celles de la situation ; c'est sur ce différentiel de certitude que s'est bâtie l'informatique comme profession[82]. *Raisonner avec les machines*, c'est ainsi découper, au sein de la situation, des ensembles phénomènes qu'on reconnaît être régis par des systèmes de règles. Au sein de ces systèmes, la réflexion peut se développer et réaliser des inférences de manière assurée. Cette possibilité est *la même*, que la machine soit naturelle ou qu'elle résulte d'une construction humaine préalable. Cette forme de raisonnement n'est donc pas spécifique au savoir « technique ». C'est bien au contraire parce qu'elle est la forme même du savoir scientifique (la systématicité) qu'elle a une si grande effectivité ; et ce n'est donc

80. Voir plus haut, § 47.
81. Voir notre note sur la notion de formalité, § 45.
82. Nous reprenons ici les conclusions de notre chapitre 3, section 3.4.

pas un hasard si science et technique se sont rencontrées dans la *mécanique* et dans l'idée de *machine* ; il est peut-être possible d'avancer que c'est à partir de ce concept commun, qui est d'abord une manière très spéciale d'appréhender l'expérience, qu'elles ont commencé leur fructueuse collaboration.

Il est important de noter que nous n'opposons pas ici la certitude mathématique à la certitude empirique ; car la machine est un dispositif empirique elle-même, et c'est *en son sein* que se noue la division du travail entre la vérification matérielle des significations et la réflexion mathématique (procédurale) qui s'en trouve par là libérée. Nous avons défini la machine[83] comme un phénomène dont la dynamique des interactions avec la situation *semblait* réglée. Le caractère modal de cette définition ne signifie pas qu'il n'y serait question que d'apparences ou de conjectures, mais bien plutôt que *juger* qu'une chose est une machine, c'est se donner licence de réaliser à son propos toutes les compositions d'inférences décrites par le système qui lui est prédiqué. C'est ce jugement initial qui est toujours empirique, quelles que soient les vérifications qui ont pu être préalablement faites, et même leur rigueur systématique éventuelle. Il n'est pas question ici de probabilité ou de « rayons cosmiques épars » mais seulement du statut du jugement, qui porte toujours *sur l'expérience*, et dont la certitude ne peut donc jamais aller au-delà de celle dont sont passibles les phénomènes eux-mêmes.

Ainsi, l'assurance qu'a le maître d'ouvrage que la machine fonctionnera doit être divisée en deux parties. Il doit d'abord *faire confiance* au programmeur, qui lui donne une *garantie* de validité de son programme. Une telle condition peut être supprimée, car le maître d'ouvrage pourrait très bien être le programmeur lui-même, ou vérifier par lui-même le programme dans son entièreté. Mais il doit faire également confiance à la machine qui est programmée, ainsi qu'à toutes ses conditions matérielles de fonctionnement (électricité, température, sécurité *vs* intentions malveillantes, etc.). C'est la nature de ces conditions *empiriques* d'effectivité de la machine que nous aimerions à présent approfondir.

On peut prendre ici pour point de départ l'assertion quelque peu énigmatique de Marvin Minsky, que nous avions déjà relevée :

> Peut-être pourrait-on même soutenir l'idée que la croyance en un énoncé arithmétique équivaut à celle que certaines machines, correctement construites, fonctionneront[84].

83. Voir plus haut, § 204.
84. (Minsky 1967, p. 5). Voir plus haut, p. 793.

L'idée de Minsky nous semble pouvoir être illustrée comme suit : croire que $2+2=4$, par exemple, équivaut à croire que $10_b + 10_b = 100_b$, en écriture binaire ; et la croyance en cet énoncé à son tour équivaut à celle que

$$\begin{aligned} xor(0,0) &= 0 \\ xor(xor(1,1), and(0,0)) &= 0 \\ or(and(1,1), and(xor(1,1), and(0,0))) &= 1 \end{aligned}$$

où $and(a,b)$, $or(a,b)$ et $xor(a,b)$ désignent respectivement les fonctions logiques ET, OU inclusif, et OU exclusif. Or ces formules peuvent être reformulées à leur tour afin de ne faire apparaître que notre fonction $f(s,b)$, et représenter alors un circuit de transmission dans notre système d'irrigation, c'est-à-dire une machine[85]. Grâce à la transcription des règles de l'addition dans ces différents systèmes de signification, la croyance en $2+2=4$ équivaut donc à celle que le circuit d'irrigation ainsi décrit, si ses canaux sont bien creusés et si ses barrières se lèvent et se baissent comme prévu, fonctionnera.

Tous ces énoncés semblent ne relever ici que d'un seul et même principe : « *Si vous comprenez et acceptez le sens de la règle, alors étant donnée sa condition, vous devez admettre sa conséquence* ». Dans le cas de $2+2=4$ et de $10_b + 10_b = 100_b$, il s'agit des règles de l'addition en nombres arabes décimaux ou binaires. Dans le cas du circuit d'irrigation, il s'agit des règles d'écoulement de l'eau d'un canal à l'autre et de son arrêt par une barrière.

On peut objecter que ces règles *ne sont pas de même nature* : alors que les premières sont simplement *stipulatives* – elles définissent ce qu'est *additionner*, les secondes sont empiriques, ou en tous les cas dépendent de lois naturelles comme celles de la gravitation et de l'écoulement des fluides. Les premières sont *appliquées* par le sujet agissant, les secondes *constatées* par lui, et dépendent donc d'une condition externe.

Les choses ne sont cependant pas aussi simples que cela. D'abord, il n'est pas possible de comprendre cette opposition comme celle qui existerait entre des *règles procédurales* et des *règles matérielles*. Effectuer une addition est une action matérielle qui peut échouer comme n'importe quel autre dispositif. Babbage déclarait que l'avantage d'une machine serait de diminuer le taux d'erreur de ses calculateurs humains. Par ailleurs, les règles de fonctionnement du système d'irrigation ne sont pas empiriques au sens où elles dépendraient d'hypothèses à faire concernant

85. Voir plus haut, § 206.

l'écoulement de l'eau, qu'elle passera dans les canaux, que les barrières l'arrêteront, etc. De telles hypothèses n'ont pas à être faites, car le flux de l'eau *se comporte déjà d'une telle façon* ; construire un circuit d'écoulement de l'eau c'est seulement en modifier un autre, plus élémentaire. La programmation, comme nous l'avons déjà dit, ne fait que modifier le contrôle *d'une machine qui tourne déjà*, ou plus précisément, encore : elle modifie la machine selon les règles même par lesquelles elle l'a *par avance* décrite – que celles-ci se révèlent correctes ou non quand l'expérience est convoquée. Qu'il s'agisse de construire un canal ou d'effectuer une addition, on peut dire que ces actions sont conformes à la signification exacte de ce qu'*est* un canal d'irrigation (un passage d'eau) et une addition ; elles ont toutes deux le même niveau de pureté et de matérialité. Que l'eau s'évapore de la surface de la terre, ou qu'une lésion cervicale atteigne mes compétences de numération, ces faits n'ont rien à voir avec la définition correcte de ces actes.

Ce n'est donc pas à ce niveau-là qu'il faut analyser la différence entre l'addition ratiocinative et celle qui est effectuée dans le circuit d'irrigation. La différence pertinente se situe plutôt au niveau de l'agentivité de la règle.

Revenons au principe que nous venons d'énoncer : *« Si vous comprenez et acceptez le sens de la règle, alors étant donnée sa condition, vous devez admettre sa conséquence »*. On peut reconnaître là le *modus ponens*. Pour être admis, il semble ne requérir qu'une forme de *cohérence avec soi-même*, car accepter une règle, c'est accepter de l'appliquer. Or il est utile de se rappeler ici les difficultés que la tortue fait à Achille à ce sujet : si elle consent à *accepter* cette *méta-règle*, à la demande d'Achille, elle refuse cependant de l'*appliquer* sans l'aide d'une autre[86], et enclenche ainsi une régression à l'infini.

Le cas de la machine contourne cette difficulté. La règle de l'addition, transcrite dans un circuit d'irrigation, n'a plus à être appliquée ; car le flux de l'eau donne directement son résultat ; il se substitue à l'effort de l'investigateur « poussant » les symboles l'un après l'autre vers leur prochaine transformation. Il semble qu'Achille se débarrasserait vite de la tortue avec une telle machine.

Cependant, une autre condition apparaît sous la plume de Minsky : « si la machine est correctement construite ».

Cette condition est nécessaire parce que l'acte de l'investigateur qui permet de rendre la règle-objet (ici l'addition) ainsi effective est *indirect*, et externe à elle ; construire ou modifier une machine obéit à des règles qui ne sont pas celles de

86. (Carroll 1895).

la règle-objet ; il faut savoir correctement creuser un canal et construire un mécanisme de barrière automatique afin que la modification du circuit soit effective.

C'est là que la tortue pourrait à nouveau faire des difficultés à Achille, de manière bien plus directe et évidente que pour le *modus ponens* : car il y a là un découplage effectif et visible entre la construction de la machine et son fonctionnement. Si elle n'est pas construite selon des règles, il n'y a aucune raison de lui accorder confiance. On a alors affaire à un dispositif naturel contingent auquel on décide de confier nos ratiocinations. Si elle l'est, alors il faut accepter ces règles et reconnaître leur application correcte. Il se pourrait que l'application de ces règles soit entièrement automatisée (dans notre exemple d'irrigation, on peut imaginer des foreuses creusant les canaux selon un programme). Mais si on peut valider le programme, encore faut-il avoir confiance dans les foreuses elles-mêmes, et ainsi de suite. Or cette imbrication de machines est bien réelle et, avons-nous remarqué, en une large mesure circulaire, puisque les ordinateurs actuels sont *toujours* construits à partir d'autres ordinateurs. Ce fait est utilisé parfois par les adversaires des méthodes formelles[87] : si vérifier la correction d'un programme requiert un autre programme, alors on est pris dans une régression à l'infini ; cela n'est rien d'autre que l'argument de la tortue, transposé de la manière que nous avons indiquée.

On peut à présent apercevoir la différence qui existe entre les règles ratiocinatives et les règles mécanisées. L'effectivité des premières fait appel au *modus ponens*, qu'on peut voir comme une forme développée du principe de cohérence, appliqué *aux intentions* de l'investigateur ; s'il admet la règle, il doit admettre de l'appliquer, car c'est *cela qu'elle veut dire*.

Il se passe quelque chose de différent avec les règles mécanisées ; elles ont toujours trait à la cohérence entre les intentions de l'investigateur, mais celles-ci à présent *passent un contrat* avec un régime extérieur de faits ou de règles, *qui lui promet* l'application correcte de la règle-objet. L'investigateur doit accepter les conclusions de la machine – car c'est bien pour cela *qu'il a d'abord passé le contrat* – cependant une condition complémentaire apparaît : *pour autant qu'il fait confiance au contractant*. Comme nous l'avons dit, ce ne sont pas tant les compétences du programmeur qui peuvent ici être mises en doute, car le programme est *ce qui peut être vérifié dans la machine*, ce qui est *mathématique* en elle, mais plutôt la fiabilité de la machine elle-même. D'une certaine manière, la distinction

87. Voir la discussion de (Bringsjord 2015, p. 275).

entre programme et machine indique seulement que cette vérification n'est pas absolue, et qu'il reste un arrière-plan de faits et de règles qui doivent à nouveau être vérifiés[88]. La machine apparaît ici, sous une autre perspective encore, comme le lieu-frontière dans l'entreprise de mathématisation de l'expérience[89].

Le maître d'ouvrage doit donc croire à l'effectivité de l'application de sa règle *pour autant qu'il croit à celle de la machine*. On peut reprendre ici ce que nous disions déjà lors du développement de notre interprétation contractuelle de la programmation : la formule de Hoare $A\{\pi\}B$ peut être comprise comme affirmant que le programme π *garantit* l'inférence de A à B[90]. C'est dans cette division du travail que se constitue l'effectivité des règles mécanisées, *supérieure* à celle de règles ratiocinatives. Par ce détour complémentaire, et cette *incertitude complémentaire* qui en résulte, l'investigateur gagne en effet l'automatisation de la ratiocination, avec tous les avantages que nous avons indiqués[91]. L'effectivité du bras robotique à construire la voiture résulte de celle de milliers de petits contrats passés entre programmeurs, qui ont permis à chacun d'implémenter leurs propres règles sur les machines de leurs confrères, construisant ainsi cette machine sophistiquée par *confiance* dans l'effectivité de machines toujours plus élémentaires.

L'effectivité des machines n'est donc rien d'autre que celle d'un réseau de significations sur la base duquel elle est conçue. En tant que ce réseau est *externe* au concept de la machine, son effectivité peut en effet sembler *empirique*, et on peut être tenté d'évoquer l'énergie, l'ingénieur, la technologie nécessaires à sa réalisation comme sources de cette effectivité. Cependant, si on prend garde qu'on ne peut concevoir des machines élaborées qu'au sein de *systèmes techniques*, on voit mieux que leur effectivité repose toute entière sur celle d'un *système de significations* qui organise une partie de l'expérience. Les machines sont effectives parce qu'elles sont conçues sur la base d'un système extrêmement complexe d'inférences contrôlées à propos de l'expérience, qui fut construit progressivement et de manière cumulative.

88. Voir plus haut, p. 174, et également § 60.
89. Voir plus haut, sections 7.4 et 7.5, § 185, § 203.
90. Voir plus haut, § 56.
91. Voir plus haut, section 11.2.

§ 245. Les détours productifs de la connaissance

La mécanisation apparaît donc être le geste par lequel on *confie* certaines conditions de la cohérence de nos intentions à des tiers, et par lequel un monde industriel peut apparaître. Elle s'ancre donc dans le même principe de cohérence des intentions qui caractérise la résolution de problème tout entière, qu'elle soit délibérative, ratiocinative ou systématique.

Or cette idée est paradoxale. On semble ici associer l'effectivité de la connaissance (sa capacité à engendrer des effets dans le monde) à la tautologie qui est souvent prise, par les détracteurs de la logique, comme emblème de sa vacuité. Pour conclure cette réflexion, nous aimerions approfondir ce qui permet ici à la simple cohérence avec soi-même d'être *productive*.

Lorsque, au lieu d'agir face à une situation, j'énonce une règle qui décrit mon action comme impliquée par la situation, et que j'applique ensuite cette règle, il semble que je n'ai fait là qu'un détour « qui revient au même ». Il semble qu'il en est de même lorsque, au lieu d'énoncer simplement une procédure, je décris tout un système de règles dans lequel cette procédure apparaît comme une composition particulière. Pourquoi faire de tels détours, *qui reviennent au même* ?

Il faut d'abord remarquer que c'est l'investigation discursive tout entière qui peut être décrite comme un tel détour. Face à une situation problématique, au lieu de céder à l'impulsion qui réagit à certaines de ses marques saillantes – que cette impulsion soit organique ou acquise par l'habitude –, les personnes ont la capacité proprement humaine d'utiliser des signes institués pour mettre à distance cette situation et *contrôler* cette impulsion afin d'examiner ses conséquences, ainsi que d'autres options possibles[92]. Il s'agit là du détour fondamental dont les deux autres ne sont que des excroissances. Énoncer une règle, c'est « répéter » mentalement l'action avant de la réaliser, et c'est donc se donner l'occasion de vérifier de manière préalable toute une série d'inférences possibles à son propos, qui vont la confirmer ou l'infirmer. Énoncer une règle au sein d'un système de règles, c'est de plus se donner d'emblée tout un ensemble de vérifications déjà établies, et si ce n'est pas suffisant, tout un arsenal de procédures de vérification *a priori* augmentant encore plus l'effectivité de la délibération.

Ces détours que sont l'énoncé des règles et la construction des systèmes sont donc des « investissements » qui permettent à l'investigation de résoudre un pro-

92. Voir plus haut, § 170.

blème une fois pour toutes, et de se contenter ensuite de s'assurer que les conditions de ce problème s'appliquent à la situation. Nous avions rencontré cette idée d'abord chez Ryle, qui avait comparé la construction des théories à l'entreprise d'un agriculteur pour terrasser les chemins qui lui permettront de mieux circuler dans ses champs[93]. C'est la même idée qui permet à Aristote de définir la technique comme « une disposition productive, accompagnée de raison vraie, de choses contingentes[94] ». Agir techniquement, c'est construire une solution générale à des problèmes dans lesquels sont inscrits une certaine variabilité ; c'est comme nous l'avons dit, concevoir l'universalisation possible de certaines délibérations, et c'est en cela que consiste sa *vérité*.

Ainsi, le principe de cohérence qui guide l'application des procédures et des systèmes face à des situations est tout simplement qu'*ils ont été conçus pour cela* : reconnaître un certain problème dans une situation, et refuser d'appliquer les solutions spécialement conçues à son égard, serait (toutes choses égales par ailleurs) se contredire soi-même, comme la tortue refuse d'appliquer le *modus ponens*.

Ainsi la mécanisation, que nous venons de caractériser comme l'échange d'une partie de notre capacité à vérifier la cohérence de nos intentions pour le bénéfice d'effectivité apporté par les machines, apparaît ici comme le cas particulier d'un schéma plus général, qui commence dès la délibération. Comme le dit Dewey lui-même, les significations du langage ordinaire sont eux-mêmes les premiers outils, ou encore, comme nous l'avons avancé nous-même, les première machines inférentielles auxquelles se *fient* nos délibérations[95]. Quant à nos procédures, elles se fient également au réseau d'outils (techniques, institutionnels, etc.) qu'elles trouvent disponibles dans notre monde – nous pensons ici au *Zeug* de Heidegger[96]. La mécanisation, qui ouvre le champ des investigations systématiques, apparaît dans cette perspective comme la troisième modalité d'une même et unique relation d'échange entre la pensée investigatrice et le monde. C'est en effet l'investigation qui fait apparaître dans le monde des réseaux de signes, d'outils et de machines ; et c'est grâce à ceux-ci qu'elle peut en retour se développer et de gagner en effectivité.

Qu'est-ce qui rend ces détours « productifs » ? D'abord il faut remarquer qu'ils ne le sont pas toujours. Il est des cas où nous échouons à énoncer des règles suffisamment « productives » pour qu'elles soient préférables à la délibération directe

93. Voir plus haut, p. 365.
94. Voir plus haut, p. 399.
95. Voir plus haut, § 142. Sur Dewey, voir plus haut, § 170.
96. *Sein und Zeit*, § 15. Voir l'article de Courtine (1987) à ce sujet.

dans des cas qui nous semblent cependant similaires. C'est par exemple ce qu'Aristote observait des situations concernant la vie humaine, et qui reste encore très largement vrai, malgré les progrès de la systématicité rappelés ci-dessus ; c'est aussi ce qui se passe avec les systèmes dits « chaotiques » ou « ouverts ». Ce qui rend ces détours productifs est donc *que la réalité s'y prête*. En quel sens interpréter une pareille formule revient à la métaphysique. On peut la comprendre par exemple de manière réaliste (puisqu'il faut bien que la situation ait une certaine structure pour que cette productivité ait lieu), ou encore idéaliste, ou encore transcendantale comme Kant. Mais il nous importe particulièrement qu'elle soit compatible avec la tradition du scepticisme – la philosophie minimale qu'une épistémologie positive doit chercher à convaincre. Or le scepticisme, tel que l'expose Sextus, admet tout à fait les vérités du sens commun, « utiles à la vie », dans le prolongement desquelles se trouvent les techniques[97].

Là s'arrête notre réflexion. Elle peut être utile à la métaphysique, si elle contribue ainsi à dégager un vaste champ épistémologique *positif*, celui de la résolution de problème, qui est ainsi *soustrait* de son investigation. De la même manière que la métaphysique ne discute pas du *contenu* de la logique, des mathématiques, des sciences, des techniques, mais seulement de la *nature* de la vérité qu'elles atteignent, cette soustraction n'est pas une perte, mais au contraire une clarification de sa question, puisqu'elle lui dégage l'horizon très large du *sens* des pratiques humaines rationnelles.

13.4 Face aux systèmes de règles

De nombreuses questions éthiques et politiques sont engendrées par les phénomènes que nous avons décrits dans ce travail. L'effectivité de la programmation peut devenir *puissance* relative de certaines groupes humains sur d'autres, puisqu'elle peut, à l'aide de machines, transformer des situations de travail, d'éducation, de culture, de guerre, d'information et de communication. Elle transforme déjà effectivement l'espace public tout entier. Qui *contrôle* la programmation est en mesure d'accroître sa propre puissance.

Les conséquences de tout cela sont immenses et nous ne pouvons prétendre en donner ici même un aperçu, et encore moins émettre des jugements ou des recommandations à leur propos. Que vont devenir le travail, le loisir, les échanges entre

97. Voir à ce sujet (Marchand 2015).

les personnes ? Comment va évoluer la distribution des richesses et des opportunités ? et l'équilibre politique et géopolitique des pouvoirs ? Que doit-on souhaiter et redouter ? Que doit-on faire – individus, communautés locales, entreprises, associations, États ?

Notre travail peut aider seulement à *clarifier* ces questions, c'est-à-dire à les poser en de bons termes. Il nous semble ainsi que deux niveaux de réponses peuvent être distingués. Au premier niveau, il s'agit de *garder le contrôle* des systèmes de règles interactifs que l'informatique permet de mettre en place. Cela peut s'entendre en un sens individuel (garder le contrôle de ses propres interactions avec ces systèmes) et collectif (réguler ces systèmes afin qu'ils soient conformes aux lois, et plus généralement les orienter dans la direction de la volonté générale). Nous n'émettons pas ici de recommandation politique sur ces sujets, mais pointons vers deux conditions essentielles pour l'effectivité de ce contrôle : d'une part que l'éducation des personnes leur permette d'acquérir une capacité critique, voire analytique concernant ces systèmes avec lesquels ils interagissent de manière toujours plus intime (§ 246), d'autre part que le droit acquière de nouvelles formes d'effectivité adaptées aux règles interactives, d'autre part (§ 247).

Cependant, on peut être sceptique quant à la capacité de telles formes d'actions à résoudre les problèmes structurels décrits au chapitre précédent, leur prolifération incontrôlée, leur opacité, la désintermédiation, etc. Nous examinons donc un second niveau possible de réponse, celui des stratégies de *déconnexion*. Nous tentons de montrer la difficulté interne auxquelles doivent faire face ces stratégies – parce que justement la technique procède de la simple injonction d'être *cohérent avec soi-même* dans ses intentions (§ 248).

§ 246. Apprendre à programmer

La première question concerne la régulation *par chaque personne*, de sa propre interaction avec les systèmes de règles interactifs et de leurs effets : accaparement, insécurité, biais de perception et de décision, etc. Comment faire pour rester *libre* face à ces règles qui nous enserrent de plus en plus ?

Notre réponse n'est pas très originale : la condition élémentaire de la liberté vis-à-vis des systèmes de règles est la compréhension des principes de leurs logiques internes. C'est en quittant le point de vue de l'utilisateur et en prenant celui du programmeur qu'on commence à se poser des questions à leur sujet, à discuter leur logique, leurs dangers et les moyens de les contourner. C'est ainsi que chaque

citoyen peut également participer au débat public concernant la régulation de ces systèmes (que nous abordons au développement suivant). Enfin, comprendre les rouages d'un système d'information* devient, dans bien des métiers, une condition de progression de carrière ; il est donc également question, ici, d'égalité des chances.

Or comprendre les ordinateurs n'est pas si difficile que cela, paradoxalement. Nous ne voulons pas nier l'extraordinaire technicité de ce sujet. Mais cette technique est aujourd'hui largement ouverte au curieux, du fait de la culture spécifique du hacking* et de l'open-source* qui y règne. Internet a ainsi permis l'éclosion de forums où des personnes partagent sans compter leurs savoirs et leurs expertises. Il suffit donc d'y être intéressé et de savoir *quelles questions poser*.

Comment éduquer les personnes (citoyens, consommateurs, salariés, etc.) à ce savoir minimal qui permet de prendre conscience de l'*épaisseur* de ces outils qui forment à présent leur quotidien ? Cet enjeu a pris de l'ampleur dans les débats pédagogiques depuis les années 2000, jusqu'à émerger de temps à autre dans le débat public. Il n'est pas de l'objet de ce travail de faire un état des lieux des diverses approches et méthodes possibles. Nous pouvons seulement pointer vers la complexité de la question.

D'un côté, l'approche la plus directe consisterait à apprendre à chacun à programmer, et cela dès son plus jeune âge. C'est la voie ouverte par Seymour Papert dès la fin des années 1960, avec la création du langage Logo[98], dont les langages visuels d'aujourd'hui sont les descendants. Programmer de manière régulière pendant un certain nombre d'années des problèmes variés, dans des langages différents, semble être la meilleure manière d'acquérir le *bon sens informatique* de Gérard Berry[99], c'est-à-dire une attitude critique relativement au comportement d'un système informatique quelconque, pouvant se muer lorsque de besoin en une posture analytique d'une complexité croissante – les programmes étant organisés de manière modulaire, il n'est en général pas difficile d'en comprendre les principes, et progressivement les détails. Un argument supplémentaire en faveur d'une telle stratégie est que la programmation peut être une façon ludique d'apprendre à résoudre des problèmes de manière structurée, et donc être utile également à l'apprentissage des mathématiques.

Le défaut de cette « voie royale » semble être son coût et donc son élitisme.

98. (Papert 1980).
99. Voir plus haut, p. 11.

Équiper des écoles d'ordinateurs qu'il faut renouveler régulièrement, former des professeurs souvent peu enthousiastes, avoir des groupes d'élèves de taille assez réduite pour les encadrer efficacement dans ces travaux pratiques – toutes ces difficultés semblent en faire une voie étroite, réservée aux écoles les plus riches. Vouloir standardiser cette approche afin de réduire son coût, ainsi que le fait avec succès le programme Hour of Code, dispensé en ligne, se fait au détriment de la complexité des exercices : la focalisation est centrée sur le simple codage, et non sur la résolution de problème[100].

D'un autre côté, la Commission Européenne prit, vers les années 1990, une posture inclusive du sujet, en visant la diffusion des simples *compétences numériques*, décrites selon différents niveaux de maîtrise. Un tel concept, construit à partir du savoir-faire de l'*usager numérique* (utiliser un traitement de texte, envoyer un courrier, utiliser un tableur, installer un programme, etc.) est orienté vers la diffusion large d'une culture numérique minimale, et donc vers la minimisation de la fracture numérique. Une telle approche ne peut néanmoins nullement former les personnes à une attitude critique et analytique face aux systèmes informatiques, car ils manquent des cadres conceptuels adéquats pour savoir *lire* leur comportement.

Une voie médiane est tentée depuis une quinzaine d'années, appelée *pensée computationnelle* (*computational thinking*)[101]. Nous avons déjà évoqué ce courant, dont l'intérêt est de reconnaître que les concepts informatiques dépassent le champ strict de l'informatique et peuvent être appliqués *directement* à la résolution de problème dans de nombreux autres domaines scientifiques et techniques. En insistant moins sur la programmation proprement dite que sur la résolution de problème, elle est plus flexible et donc peut sans doute mieux être diffusée dans des milieux socio-économiques différents. Son problème semble être de garantir la qualité de l'enseignement, l'idée de *pensée computationnelle* pouvant devenir un fourre-tout conceptuel, difficilement évaluable ; de nombreuses activités se réclamant d'elle n'ont qu'un rapport assez élusif avec l'informatique, et beaucoup d'enseignants les proposent à leurs élèves sans savoir ce qu'on doit en tirer[102]. La clarification préliminaire de *ce qu'est* un programme, et la mise au jour de la complexité de cette notion, est à ce titre une tâche essentielle pour la définition des

100. (Lonati, Malchiodi et al. 2015).
101. (Wing 2006). Voir plus haut, § 10.
102. Échange personnel avec Violetta Lonati, septembre 2023.

bonnes stratégies éducatives. Les projets de Lonati, Brodnik et al. (2022) et de l'ANR PROGRAMme y participent pleinement. Notre travail vise également à y contribuer, en ce qu'il explique la prégnance de l'informatique dans notre monde par le rôle assez élémentaire que joue la programmation dans l'économie de notre connaissance pratique – de notre pouvoir d'agir ; il peut contribuer par là, nous l'espérons, au « décloisonnement » de cette discipline et à la compréhension de son importance hors des cercles scientifiques et techniques.

§ 247. La régulation des règles

À l'échelle collective, *régulation* ne signifie pas seulement *droit du numérique* (protection des données et de la vie privée, propriété numérique, vente en ligne, etc.). Il s'agit de savoir quels régimes de règles appliquer à ces nouvelles institutions : Les réseaux sociaux sont-ils vraiment des *média* ? Les places de marché ne sont-elles que des entreprises de commerce ? Les nouvelles monnaies remettent-elles en cause le monopole traditionnel de l'État sur la monnaie ? etc. Comment par ailleurs corriger des asymétries de pouvoir avérées ou potentielles entre individus (consommateur, citoyen, travailleur, etc.), les intérêts privés, et les intérêts publics ? En particulier, quel équilibre de pouvoir souhaiter entre les entreprises multinationales de technologie et les États (et ici il ne faut pas seulement penser aux démocraties libérales, mais aussi aux démocraties illibérales et aux régimes autoritaires) ? etc.

Ce sur quoi nous voulons porter l'attention ici – puisqu'il n'est pas question d'entrer dans le vif de ces questions politiques – est leur *modalité* : *réguler des systèmes de règles*. On pourrait hausser les épaules, et remarquer que le législateur ne fait presque jamais que cela, réguler des institutions ou l'emploi de technologies. Cependant ici il n'est pas question de réguler seulement des institutions ou des technologies particulières mais, comme nous avons tenté de le montrer, de mettre sous contrôle *un changement généralisé dans la manière dont tous les systèmes de règles deviennent effectifs*.

Pour donner un seul exemple, très élémentaire, la notion de *responsabilité* se trouve profondément reconfigurée par le fait que ces systèmes de règles agissent *de manière autonome* : le vieux principe de la *responsabilité du fait des choses*, qui veut que les dommages causés par une machine soient imputables au *gardien* de celle-ci, *i.e.* son propriétaire ou la personne qui la dirige effectivement, est mis à mal car trop aisément opposable ; le cas d'un accident causé par un véhicule auto-

nome dans une situation ambiguë permet de se représenter aisément la difficulté. Le Parlement européen a ainsi proposé, de manière controversée, de créer à long terme une personnalité électronique pour les robots[103].

Il ne s'agit pas seulement d'adapter des concepts juridiques à cette nouvelle réalité, mais de comprendre la manière dont le droit pourrait *lui-même* devenir effectif sur ces questions, c'est-à-dire la manière dont une *police des règles* pourrait appliquer les régulations portant sur les systèmes de règles directement effectifs.

Le très éclairant ouvrage de Garapon et Lassègue, *Justice digitale*, montre comment l'*immédiateté* du numérique (par exemple celle des contrats intelligents* déjà évoqués[104]) défie la lenteur intrinsèque de la justice, qui requiert l'écoute, le débat, la réflexion, parce que *justement* l'événement doit être mis à une certaine distance du juge afin qu'il puisse porter sur lui un regard non biaisé. En contrepartie, cette lenteur permet au droit d'avoir un approche souple et *non systématique* des affaires humaines. Des cas imprévus et nouveaux ne sont jamais un problème pour un tel système de règles, puisque le juge est justement là pour les traiter, et éventuellement faire jurisprudence. Droit et informatique sont donc dans un conflit assez élémentaire de *logiques différentes* concernant les règles.

L'immédiateté du numérique exige qu'un contrôle *a priori* des systèmes de règles soit effectué, ce qui pourrait prendre la forme, non seulement de règlements auxquels ils devraient se soumettre par une certification préalable, mais également *d'une administration et d'une police automatisées* (à défaut d'un meilleur terme) intervenant directement dans les différents systèmes objectifs interactifs. Comme l'écrit de manière presciente le professeur de droit Lawrence Lessig[105], « Le code fait loi » (*Code is Law*), c'est-à-dire que les programmes informatiques définissent *de facto* ce qu'il est licite de faire ou non concernant les interactions qu'ils contrôlent ; simplement Lessig limitait ce principe au *cyber-espace*. Nous avons tenté de montrer, au chapitre précédent, qu'il était valide pour toutes les pratiques humaines, à proportion de l'informatisation de leurs techniques et de leurs institutions.

Dans les États autoritaires (par exemple la Chine), la surveillance des réseaux sociaux est ainsi *directement* réalisée par des systèmes mis en place par l'État, auxquels tous les fournisseurs de services numériques doivent donner accès. Dans

103. *Résolution du Parlement européen*, 16 février 2017, procédure n° 2015/2103(INL). Voir à ce sujet le commentaire de (Hubin 2018).
104. Voir plus haut, § 229.
105. (Lessig 2000).

les démocraties, on délègue la modération des réseaux sociaux à leurs opérateurs, qui doivent mettre en place eux-mêmes, au sein de leurs propres systèmes, les règlements imposés par le législateur, par exemple concernant les discours haineux ou la discrimination. De la même manière, les banques doivent elles-mêmes mettre en place des mécanismes de contrôle des flux financiers qu'elles hébergent, afin de détecter les blanchiments d'argent. Cependant, l'État numérique s'intègre de plus en plus aux systèmes privés, par exemple à propos des enjeux de sécurité nationale[106] ou, dans un domaine plus civil, à propos de la collecte automatique de la TVA dans les transactions en ligne[107]. De la même manière, les monnaies numériques comme le BitCoin sont en voie d'être régulées.

On voit donc que réguler les systèmes objectifs interactifs requiert de s'intégrer à eux, ce qui ajoute des règles aux règles. De ce fait les questions politiques sont *aussi* des questions techniques, car il n'est plus seulement question de capacité de coercition sur des individus, mais également d'intervention directe sur des machines et sur toutes les strates de programmation accumulées par dessus elles. On peut formuler cela de manière plus large encore en disant que le numérique a tendance à transformer les règles institutionnelles en règles techniques – à techniciser le droit et la politique. Un exemple est le cas, fameux en France, des algorithmes APB et ParcourSup, qui ont entériné l'automatisation du processus d'accès aux études supérieures. Le débat politique devient alors difficile car il porte à présent sur des critères techniques, voire mathématiques de l'algorithme, comme par exemple l'effet du rôle asymétrique assigné aux universités et aux étudiants dans l'implémentation de l'algorithme de Gale-Shapley[*], ou du recours à l'aléatoire[108]. *De facto*, on retrouve là la désintermédiation déjà évoquée[109], qui met l'humain *à la périphérie* des processus institutionnels.

C'est ainsi que Garapon et Lassègue terminent leur ouvrage par une mise en garde ferme contre la tentation pour le droit de croire que son *intégration* aux systèmes de règles interactifs sera en mesure de les orienter en direction de la volonté générale :

> [La] justice digitale […] recherche des automatismes et des règles s'appliquant sans discussion. […] Elle est à l'image des phénomènes qui accompagnent la révolution graphique que sont la mondialisation, l'économie et la gouvernementalité néolibé-

106. (Sarri et al. 2022).
107. (Giacomo et al. 2022).
108. (Pégny à paraître, chap. 3.4).
109. Voir plus haut, § 231.

> rale, lesquels ont tous un fonctionnement en système : ce sont des formes englobantes, hostiles à toute extériorité. La rhétorique de tous ces systèmes, qu'ils soient économique, numérique ou mondialisé, est de renforcer autour d'eux une clôture (« Il n'y a pas d'autre choix » ; « There is no alternative »). Ils n'excluent pas d'intégrer des principes de justice, comme en témoignent la responsabilité sociale des entreprises (RSE), la lutte contre la corruption ou l'écologie, mais ceux-ci n'ont de place qu'à la condition de se couler dans le système englobant, ce qui lui permet d'échapper à tout jugement extérieur et de n'être jamais interpellé[110].

Aussi se montrent-ils sceptiques relativement à une police des règles automatisée. Ils insistent sur l'importance de maintenir certaines des formes matérielles et symboliques de la justice actuelle, et notamment ce qu'ils appellent le moment de l'audience, où a lieu la confrontation entre les personnes, « un lieu hors de la vie courante dans lequel la résolution de conflits ne passe plus par la violence réciproque mais seulement par la parole et l'argumentation, [à la] temporalité cérémonielle[111] ». La justice numérique aura beau montrer l'inefficacité d'un tel dispositif, sa lenteur, sa faillibilité, et son remplacement possible par des procédures de résolution automatisées ou semi-automatisées, Garapon et Lassègue insistent *justement* sur cet arrêt, cette *interruption* que doit constituer le moment de justice dans le tourbillon des systèmes dynamiques de règles.

§ 248. La déconnexion

La recommandation de Garapon et Lassègue, de maintenir des moments fermes d'*interruption* des logiques numériques afin que les personnes se retrouvent directement, face à face, pour traiter leurs différends, est représentative d'une posture radicalement distincte de la tentative de « réguler les systèmes de règles », que nous appelons ici du terme générique de *déconnexion*.

Cette posture peut en particulier être inspirée par les critiques modernes du phénomène technique (Ellul, Heidegger, Marcuse, etc.) qui en ont montré le caractère englobant et autonome, encerclant l'être humain plutôt que se plaçant à son service. Nous avons nous-même relevé l'aspect systémique de l'automatisation[112].

Face à cette emprise, sont donc envisagées des stratégies de « déconnexion », de

110. (Garapon et Lassègue 2018, p. 329).
111. (ibid., p. 180).
112. § 230, § 231.

retrait hors de ce tourbillon de règles interdépendantes, impliquant asservissement et vulnérabilité. Il ne s'agit plus ici de maîtriser ou de négocier les règles à l'intérieur du système, mais de *résister* à celui-ci. Cette résistance peut se concevoir comme totale ou circonscrite (par périodes de temps, par types de services, etc.), elle est en tous cas disciplinée[113].

On observe ces stratégies d'abord au niveau individuel. Elles sont signifiées aujourd'hui par le refus emblématique du *smartphone* – et elles ont en général un sens également politique[114]. Mais il existe également des déconnexions réalisées par les institutions elles-même, comme lorsque des entreprises décident de boycotter une plateforme de vente en ligne, ou lorsqu'un État interdit un réseau social à l'échelle du pays tout entier (Facebook en Chine). En Inde, les États régionaux qui ne disposent pas de compétences numériques suffisantes n'hésitent pas à déconnecter Internet tout entier pendant quelques jours lorsqu'ils craignent des campagnes de désinformation en ligne[115].

Ces stratégies de déconnexion se qualifient parfois elles-mêmes de *marginalisation*. Elles sont le pendant de la *fracture numérique*, qui désigne le problème sociétal d'une déconnexion *subie*. On reconnaît par là que nos systèmes de règles objectifs, techniques et institutionnels, c'est-à-dire l'infrastructure de nos sociétés, sont pour une part significative déjà informatisés, et qu'être exclu de ce monde numérique, c'est être exclu tout court. Pour les individus, la déconnexion a d'ailleurs souvent pour corollaire un sentiment d'anxiété, le célèbre *FOMO*[116].

La difficulté de telles postures de résistance est que cette inflitration ne procède pas, fondamentalement, de *nouvelles* activités proposées aux personnes qui viendraient ainsi les détourner de l'essentiel. C'est au contraire les *mêmes* pratiques sociales et culturelles qui changent de modalité. Même si les personnes passent de plus en plus de temps en ligne, elles ne sont pas vraiment *ailleurs* – sauf peut-être lorsqu'elles jouent, mais cela tient au fait même de jouer, et non à sa modalité numérique. Elles sont avec d'autres personnes, elles travaillent, s'informent, étudient, règlent des tâches administratives. Comme le dit Jauréguibbery, c'est d'abord une logique « de l'intégration et de l'appartenance[117] » qui pousse les personnes à res-

113. Sur le phénomène de *déconnexion* voir les études de Francis Jauréguiberry, en particulier (Jauréguiberry 2014).
114. (Gavard 2023).
115. Economist, juillet 2023.
116. *Fear of Missing Out*, « peur de rater quelque chose ».
117. (Jauréguiberry 2014, p. 21).

ter connectées.

La modalité systématique par laquelle le numérique leur enjoint de faire les choses est certes partiellement subie. Cependant elle est aussi, pour une plus large part encore, volontaire ; il y a ici une deuxième logique qui est à l'œuvre, celle « de l'efficacité et de la performance[118] ». Le numérique permet aux personnes de faire plus vite, mieux, moins cher, ce qu'elles voulaient *déjà* faire : envoyer un courrier électronique plutôt qu'un courrier postal, déclarer ses impôts en ligne plutôt que sur papier, s'informer sur Internet plutôt que dans les bibliothèques, etc. Il permet de reconfigurer toutes ces tâches selon leur stricte forme essentielle, comme nous avons tenté de le montrer[119]. Faire ces choses ainsi c'est, pour les personnes qui y adhèrent, simplement *mieux* les faire, parce que la manière et les moyens sont ici ajustés *exactement* à leur fin.

La difficulté, donc, de ces postures de résistance est qu'en une certaine mesure leur adversaire est la *logique* même des intentions. D'une part, il y a la logique de s'intégrer au corps social, ce qui passe par l'acceptation de ses règles ; or nous avons montré que le *régime des règles* que les machines invitaient à mettre en place n'était plus accessible sans leur intermédiaire – puisque la règle est souvent elle-même *calculée* au moment même où elle doit s'appliquer. D'autre part, il y a la logique de réaliser ses intentions le plus effectivement possible, sans surplus de moyens, de temps, d'énergie que nécessaire.

Bien sûr, au niveau individuel, ce que nous disons là ne vaut pas : il est tout à fait possible, pour de très bonnes raisons, que certaines personnes préfèrent le courrier postal au courrier électronique, et qu'elles décident de communiquer textuellement uniquement de cette manière. La difficulté consiste à concevoir la possibilité (sans bouleversement majeur et mondial) que de telles préférences se généralisent, pour une majorité de personnes et d'institutions dans le monde, pendant un *même et assez long* temps, sur l'ensemble des applications du numérique (y compris, par exemple, l'éducation et la santé), de sorte qu'une telle bascule ait effectivement lieu : car toutes ces choses sont liées entre elles, et chacune est tenue par toutes les autres. Si donc la résistance est toujours possible au niveau individuel et local, ce qu'il est difficile aujourd'hui de concevoir, c'est une stratégie globale et *offensive* de déconnexion, qui ne soit pas coercitive pour les personnes.

Afin de comprendre cela, il est utile de revenir à la pensée de Heidegger sur la

118. (ibid., p. 23).
119. Voir plus haut, § 231.

technique. L'exigence de systématicité, que nous avons tenté de rendre manifeste, ne serait qu'un autre nom de ce que le philosophe allemand appelle *le principe de raison*[120], puisqu'elle ne vise qu'à étendre, autant que possible, le domaine des règles.

La force de cette idée est de faire sentir la profondeur de ce courant souterrain, situé *bien en deçà* des objectifs et des projets particuliers des personnes et des institutions, bien en deçà de toutes les autres valeurs et même des notions de *bien* et de *mal*. Les organisations caritatives les plus irréprochables travaillent aujourd'hui de manière systématique, et donc mettent en place des règles, des procédures, et des systèmes informatiques car cela est la meilleure manière de *maximiser le bien* qu'elles veulent faire.

Ainsi, vouloir faire les choses systématiquement, ce qui passe souvent par l'aide d'ordinateurs, c'est semble-t-il, seulement *mieux vouloir* ce qu'on veut déjà. Il est important de bien soupeser la force de cette contrainte qu'impose la systématicité. Elle s'ancre dans le principe de cohérence avec soi-même, tel que nous l'avons analysé tout à l'heure[121] ; elle n'est que le déploiement d'un réseau d'équivalences strictes concernant les intentions du sujet, qui fait qu'il ne peut vouloir *ceci* sans également vouloir *cela*. Or ce qu'il veut (biens matériels, reconnaissance, sécurité, liberté, etc.) sont des choses instituées, et il ne peut donc les vouloir qu'en souhaitant également que perdurent ces institutions elles-mêmes (ce qui n'empêche pas de vouloir les réformer). Pour donner un exemple assez simple de ce que nous voulons dire : militer pour la privauté sur les réseaux sociaux ne peut être que le fait de personnes qui les utilisent et qui souhaitent que ceux-ci *continuent à exister* ; or ceux-ci, en tant qu'institutions, agrègent de nombreuses intentions de différentes parties du corps social. Être cohérent avec soi-même, ici, implique le compromis entre sa volonté propre et l'ensemble de ces autres volontés ; mais alors, on se situe d'emblée dans l'attitude précédente, celle de la politique de régulation des règles.

Aussi la contrainte imposée par le numérique apparaît-elle toujours comme indirecte. *Rien n'empêche* l'acheteur, dans l'exemple de l'ERP* déjà considéré[122], de passer commande auprès d'un fournisseur non référencé. Simplement celui-ci ne pourra pas être payé, à moins que l'acheteur obtienne d'un responsable qu'il débraye le système pour gérer cette exception. On peut *toujours*, nominalement,

120. (Heidegger [1957] 2003).
121. Voir plus haut, § 243.
122. Voir plus haut, § 51.

contourner un système informatique – ne serait-ce qu'en le réinitialisant ou en le déconnectant. C'est *le coût* de cette opération qui dissuade de le faire ; et ici le coût n'est pas un coût technique, mais un coût humain, qui se mesure en termes de désorganisation, de risques d'incompréhension ou de fraude, de temps perdu.

La contrainte exercée sur la situation, on se rappelle, est la seconde grande classe des « avantages » offerts par l'approche systématique des pratiques[123]. Nous voyons ici le lien intime qui les lient. La systématicité consistant à vouloir *plus exactement*, la contrainte qu'elle impose à chacun est simplement d'être cohérent avec soi-même, de ne pas se dédire, de ne pas changer d'avis.

Ce qui rend cette contrainte forte, voire oppressante, est que l'exigence de cohérence n'est pas toujours naturelle à la volonté humaine : nous voulons en général *avec les autres*, et sommes habitués à affermir notre volonté grâce à leur assentiment (ou, pour les esprits contraires, à leur opposition). Nous avançons des désirs ou des projets *pour voir* comment les autres réagiront, et nous ajustons régulièrement notre volonté à des contraintes collectives. Nous savons également qu'il est parfois possible de *jouer* avec les règles collectives, de faire un pas de côté sans que cela prête à conséquence ; on sait également quand il est possible de faire soi-même exception à une règle qu'on a pu édicter quelque temps auparavant. L'immédiateté du retour de la machine change tout cela, et nous met face à tous nos engagements actuels et passés, qu'elle juge *à la lettre*, nous laissant ainsi dans l'embarras de devoir clarifier tout de suite une série de choix que nous aurions préférés garder indéterminés ou ambigus.

Cette remarque nous fait sentir combien est étrange cette injonction d'être cohérents avec nos propres intentions, que nous adressent les machines. Car ce n'est pas du tout comme cela que nous *raisonnons*. Le paradoxe *pour nous* est le suivant : comment l'injonction d'être cohérent avec nos propres intentions – de mieux vouloir ce que l'on veut – peut-elle engendrer l'impersonnalité des procédures et des systèmes ? Lorsqu'on évoque la « cohérence avec soi-même », on pense plutôt à la constance d'intention qui peut guider une personne à travers la longue durée en orientant ses choix face aux situations diverses de son existence, de sorte qu'on puisse y reconnaître, malgré les accidents et les détours de la vie, l'unité d'une histoire personnelle – son projet, diraient les existentialistes.

Ces deux interprétations de ce que peut vouloir dire *être cohérent avec soi-même* ne s'opposent pas tant qu'elles ne s'ignorent. Elles se situent simplement

123. Voir plus haut, section 5.4.

sur deux niveaux étrangers de signification, qui accordent également des sens différents à ce que veut dire *intention, soi-même, vérité, effectivité*. L'interprétation que nous avons suivie jusqu'ici traite d'intentions qu'il est possible de spécifier, c'est-à-dire de décrire indépendamment des particularités de la situation, et donc aussi de celles du sujet de ces intentions, qui devient un *soi* impersonnel, capable d'être endossé par quiconque. *Effectivité* ne veut donc pas non plus y dire la même chose ; dans la seconde interprétation, quand bien même une personne n'aurait connu que des échecs et des malheurs dans ses entreprises, on pourrait juger tout de même que sa vie aura été *effective*, si à travers ces entreprises se manifestait la constance et le courage d'une intention tendue vers un bien admirable, bien plus que celle d'une personne ayant mérité son succès par l'application de règles socialement convenues et techniquement éprouvées.

Il n'est pas ici pertinent d'aller plus loin dans cette direction. Cet exemple sert seulement à reprendre notre indication initiale, que tout ce que nous avons dit dans ce travail à propos de la connaissance, de la vérité, et de l'effectivité, ne prétend pas enserrer ces notions de manière univoque. Nos explications n'en capturent qu'un aspect précis, celui de la vérité et de l'effectivité *pratiques*. Cela ne veut pas dire que tout le réel et tout le dicible s'y réduisent. Nous tenons qu'il y a d'autres aspects de la vérité – celui de la vérité historique n'étant qu'un parmi d'autres – pour lesquels la systématicité et la ratiocination ne sont d'aucun usage.

La vérité dont relève la programmation est celle de la technique et des pratiques humaines, qui sont les lieux où leurs intentions rencontrent les choses et se rencontrent entre elles. Lorsqu'on entreprend de reconfigurer ces lieux rationnellement, ils prennent l'aspect de labyrinthes de règles, faits uniquement de l'enchevêtrement de ces intentions et de leur interactions avec le monde. Que ces labyrinthes tendent à devenir gigantesques et d'une complexité diabolique ne veut pas dire que nous devions nécessairement nous y perdre. Nous avons à présent des machines qui peuvent les parcourir à une vitesse également diabolique. L'important est de savoir qu'y chercher. Nous saurons toujours en sortir du moment que nous ne perdons pas le *fil* de nos intentions.

Annexe A

Le *General Problem Solver*

Nous introduisons et illustrons d'abord le GPS sur un exemple (§ 249). Nous montrons ensuite comment fonctionne le programme (§ 250), avant de présenter de manière détaillée une de ses implémentations possibles (§ 251).

§ 249. Introduction et illustration de l'algorithme

Le GPS est un algorithme intéressant à trois titres au moins. D'abord, il est, sinon le premier, du moins le plus emblématique des programmes de la première intelligence artificielle. Publié en 1959, il fit sensation, sans doute par sa prétention à la généralité (le G du GPS). Son abstraction lui permet en effet de traiter toute une classe de problèmes ayant une structure minimale, qu'on pourrait croire universelle. Aussi le GPS pouvait vraiment donner l'impression, en 1959, que la machine était capable de raisonner.

La seconde raison de son intérêt est qu'en effet le GPS implémente la célèbre heuristique de la fin et des moyens qu'Aristote présente en *Éth. Nic.* III.3[1]. Les procédures ainsi composées sont l'archétype de l'idée intuitive qu'on s'en fait : un plan d'actions faisant progresser par étapes séquentielles une situation d'un état initial à un état final. L'intérêt de l'algorithme réside dans son modèle de la situation, qu'il permet de représenter comme une collection d'attributs atomiques indépendants et indifférents les uns vis-à-vis des autres. Les actions, quant à elles, peuvent seulement ôter ou ajouter des attributs à la situation. Ce modèle est certes fruste, mais il est assez flexible pour permettre de de représenter de très grandes

1. Voir plus haut, § 98. L'excellent cours en ligne d'Adam Stiff, de l'université de l'Ohio, dont cette annexe s'inspire, introduit le GPS par le texte d'Aristote.

classes de problèmes. son intérêt est de montrer que c'est justement dans la représentation de la situation que se joue l'essentiel de la résolution de problème. Cette simplicité permet d'illustrer de manière particulièrement nette deux de nos thèses :

1. Les trois principes identifiés pour qu'il y ait résolution systématique de problème[2] sont très visibles : le principe de localité, d'abord (chaque opérateur dépend d'un nombre fini de conditions dans la situation), celui de composition (les actions transforment la même situation), et celui d'induction (puisque la combinatoire des situations est finie).
2. L'heuristique procède selon une logique entièrement de la procédure qui en résulte, illustrant bien la différence entre l'investigation et sa reconstruction procédurale[3].

La troisième raison de son intérêt est que, malgré des situations aussi simplement modélisées, la difficulté d'atteindre la systématicité est patente. Par exemple, le programme que nous allons présenter peut être aisément « trompé », par l'ordre dans lequel on lui présente les objectifs à atteindre.

Afin d'expliquer le fonctionnement du GPS de Newell, Shaw et Simon (1959), nous allons prendre l'exemple du chimpanzé et du régime de bananes. Le chimpanzé souhaite ne plus avoir faim (état final désiré), et au début du problème la situation est la suivante : les bananes sont accrochées au plafond au milieu de la pièce ; elles sont accessibles au chimpanzé s'il monte sur une chaise et qu'il a les mains libres. La chaise et le chimpanzé sont près de la porte et le chimpanzé a une balle en main.

L'algorithme prend pour données trois séries d'informations :

1. Une série d'états initiaux valides, ici :
   — « La chaise est près de la porte ».
   — « Le chimpanzé est près de la porte ».
   — « Le chimpanzé est au sol ».
   — « Le chimpanzé a une balle en mains ».
2. Une série d'états cibles, ici simplement : « ne plus avoir faim ».
3. Une série d'opérateurs, dont chacun est défini par quatre éléments, que nous illustrons sur un seul opérateur :
   — le nom de l'action : « attraper les bananes » ;

2. Voir plus haut, § 160.
3. Voir plus haut, § 173.

— ses préconditions, *i.e.* les états qui doivent être valides afin qu'elle puisse se dérouler, ici : « avoir les mains vides », « être proche des bananes » ;
— les états qui deviennent valides suite à la réalisation de l'action, ici : « avoir les bananes » ;
— les états qui deviennent invalides suite à la réalisation de l'action, ici : « avoir les mains vides ».

La structure complète du problème est donnée dans le programme A.2, à partir de la ligne 46.

§ 250. Éxécution de l'algorithme

L'illustration du résultat de l'algorithme est donnée de manière graphique en figure A.1.

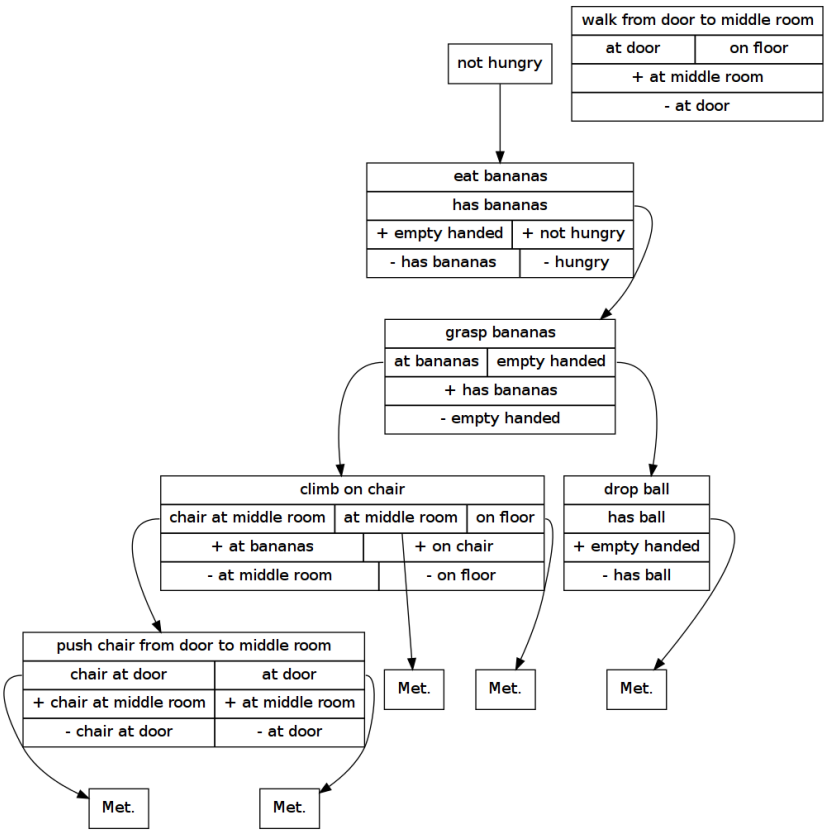


FIGURE A.1 – Illustration du GPS
Source : web.cse.ohio-state.edu

Ce qui s'affiche à l'écran lors de l'exécution du programme est donné dans le code A.1 où nous avons ajouté une indentation pour bien mettre en valeur la structure récursive de la résolution. La ligne 22 affiche le résultat souhaité, c'est-à-dire la suite d'actions entreprises pour résoudre le problème.

Code A.1 – Éxecution du programme GPS

```
1  0 Achieving: not hungry
2  0 Consider: eat bananas
3      1 Achieving: has bananas
4      1 Consider: grasp bananas
5          2 Achieving: at bananas
6          2 Consider: climb on chair
7              3 Achieving: chair at middle room
8              3 Consider: push chair from door to middle room
9                  4 Achieving: chair at door
10                 4 Achieving: at door
11             3 Action: push chair from door to middle room
12             3 Achieving: at middle room
13             3 Achieving: on floor
14         2 Action: climb on chair
15         2 Achieving: empty handed
16         2 Consider: drop ball
17             3 Achieving: has ball
18         2 Action: drop ball
19     1 Action: grasp bananas
20 0 Action: eat bananas
21
22 ['Executing push chair from door to middle room', 'Executing climb on
       chair', 'Executing drop ball', 'Executing grasp bananas', '
     Executing eat bananas']
```

Avec ces deux visualisations, on peut comprendre intuitivement comment fonctionne cet algorithme. Il part de la série d'états cibles, qu'il cherche à réaliser l'un après l'autre. Le programme affiche alors `Achieving: xxx`, précédé du nombre d'états cibles à réaliser.

Pour chacun, il tente d'appliquer successivement les opérateurs qui rendent l'un de ces états valides. Le programme affiche alors `Considering: xxx`. Les préconditions de l'opérateur s'ajoutent à la série des états cibles à réaliser. S'ils font tous partie de la série des états actuels du système, alors le programme exécute l'action (il affiche `Achieving: xxx`), modifie la série des états actuels en consé-

quence, et passe à l'état cible suivant. Sinon, il procède récursivement et cherche à réaliser les préconditions manquantes. Si une impossibilité survient (plus d'opérateur à appliquer, ou circularité) la branche en cours d'exploration est abandonnée.

§ 251. Le programme

Le programme (dont la source est donnée en ligne 2 du code A.2) est composé de quatre fonctions. La fonction principale `gps` (l. 4-11) constitue l'interface avec l'utilisateur. Elle a pour argument les trois séries de données initiales évoquées plus haut, et renvoie la série des actions menant des états initiaux aux états finaux, précédés du préfixe `Executing xxx` (« J'exécute xxx »). Dans le programme, elle est invoquée par la ligne finale (l. 108) qui lui fournit les arguments donnés par la structure de données nommée `problem`, et dont l'explicitation s'étend des lignes 46 à 87.

L'essentiel se déroule dans les trois fonctions centrales qui fonctionnent en boucle récursive. On voit que la fonction `achieve_all` (« Réalise tous les objectifs », l. 13-31) appelle, ligne 15, la fonction `achieve` (« Réalise l'objectif », l. 23-35) qui elle-même appelle, ligne 33, `apply_operator` (« Applique l'opérateur », l. 37-45). Cette fonction appelle `achieve_all` ligne 39, ce qui met en place la structure récursive de la procédure.

Nous expliquons brièvement chacune de ces trois fonctions, en ordre inverse.

La fonction `apply_operator`, donc, appelle `achieve_all` afin qu'elle considère la réalisabilité de ses propres préconditions, comme on le voit dans les arguments qui lui sont donnés ligne 39 (`operator['preconds']`). Si ce résultat est positif (test l. 40-41) alors la fonction renvoie le nouvel état du système en ôtant les états devenus invalides et en ajoutant ceux devenus valides (ligne 45).

La fonction `achieve` se concentre sur un seul objectif qui lui est donné en entrée (`goal`), et teste d'abord deux choses :

— Si l'objectif est dans l'état actuel du système, cela veut dire qu'il est déjà réalisé, et elle renvoie cet état actuel (l. 25-26)

— Sinon, et si l'objectif est également dans la série des autres objectifs restant à réaliser, cela veut qu'il y a une situation de circularité. La fonction renvoie le signal `None` (« Rien ») qui indique qu'il faut abandonner cette branche d'exploration (l. 27-28).

Dans les autres cas, la fonction `achieve` va tenter d'appliquer tous les opérateurs qui réalisent l'objectif (l. 31-32), et pour cela appelle la fonction `apply_operator` que nous venons de voir (l. 33). Dès qu'elle obtient de cette fonction un résultat positif (un nouvel état du système, qui forcément réalise son objectif), elle renvoie ce nouvel état (l. 34-35). Si tous les opérateurs possibles sont épuisés, elle se termine sans rien renvoyer.

Enfin, la fonction `achieve_all` tente de réaliser l'un après l'autre chaque objectif de la série d'objectifs assignés en mettant chaque fois à jour le nouvel état du système (l. 14-15). Si à un moment donné un objectif se révèle impossible, l'algorithme renvoie le signal `None` (l. 16-17). Une fois que tous les objectifs sont atteints, une vérification complémentaire est réalisée (l. 18-20) que tous les objectifs initiaux sont bien réalisés. En effet, il se pourrait que la réalisation d'un objectif subséquent rende invalide un objectif réalisé auparavant.

C'est ici qu'on voit la fragilité du programme : selon l'ordre dans lequel on présente les objectifs à atteindre, le programme pourra trouver une solution ou non. Pour rendre cela évident, nous avons développé un second problème (l. 89-105 du code) qui comprend deux objectifs `A,B`. L'action `a` qui permet de réaliser `A` annule l'état `B`, qui peut être réalisé par l'action `b`. Il est facile de vérifier que le programme peut trouver la séquence `a,b` si on lui présente les objectifs comme `A,B`, mais échoue si on lui présente `B,A`.

Code A.2 – Programme Python du GPS

```
1  ## General Problem Solver
2  # CREDIT TO -> Daniel Connelly @ https://github.com/dhconnelly
3
4  def gps(initial_states, goal_states, operators):
5      prefix = 'Executing '
6      for operator in operators:
7          operator['add'].append(prefix + operator['action'])
8      final_states = achieve_all(initial_states, operators, goal_states
           , [])
9      if not final_states:
10         return None
11     return [state for state in final_states if state.startswith(
           prefix)]
12
13 def achieve_all(states, ops, goals, goal_stack):
14     for goal in goals:
15         states = achieve(states, ops, goal, goal_stack)
```

```
16          if not states:
17              return None
18      for goal in goals:
19          if goal not in states:
20              return None
21      return states
22
23  def achieve(states, operators, goal, goal_stack):
24      print(len(goal_stack), 'Achieving: %s' % goal)
25      if goal in states:
26          return states
27      if goal in goal_stack:
28          return None
29
30      for op in operators:
31          if goal not in op['add']:
32              continue
33          result = apply_operator(op, states, operators, goal,
                goal_stack)
34          if result:
35              return result
36
37  def apply_operator(operator, states, ops, goal, goal_stack):
38      print(len(goal_stack), 'Consider: %s' % operator['action'])
39      result = achieve_all(states, ops, operator['preconds'], [goal] +
            goal_stack)
40      if not result:
41          return None
42      print(len(goal_stack), 'Action: %s' % operator['action'])
43      add_list, delete_list = operator['add'], operator['delete']
44      return [state for state in result if state not in delete_list] +
            add_list
45
46  problem1 = {
47      "start": ["at door", "on floor", "has ball", "hungry", "chair at
            door"],
48      "finish": ["not hungry"],
49      "ops": [
50      {
51          "action": "climb on chair",
52          "preconds": ["chair at middle room", "at middle room", "on
                floor"],
```

```
53          "add": ["at bananas", "on chair"],
54          "delete": ["at middle room", "on floor"]
55      },
56      {
57          "action": "push chair from door to middle room",
58          "preconds": ["chair at door", "at door"],
59          "add": ["chair at middle room", "at middle room"],
60          "delete": ["chair at door", "at door"]
61      },
62      {
63          "action": "walk from door to middle room",
64          "preconds": ["at door", "on floor"],
65          "add": ["at middle room"],
66          "delete": ["at door"]
67      },
68      {
69          "action": "grasp bananas",
70          "preconds": ["at bananas", "empty handed"],
71          "add": ["has bananas"],
72          "delete": ["empty handed"]
73      },
74      {
75          "action": "drop ball",
76          "preconds": ["has ball"],
77          "add": ["empty handed"],
78          "delete": ["has ball"]
79      },
80      {
81          "action": "eat bananas",
82          "preconds": ["has bananas"],
83          "add": ["empty handed", "not hungry"],
84          "delete": ["has bananas", "hungry"]
85      }
86      ]
87  }
88
89
90  problem2 = {
91      "start": ["C"],
92      "finish": ["B","A"],
93      "ops": [
94      {
```

```
95          "action": "a",
96          "preconds": ["C"],
97          "add": ["A"],
98          "delete": ["B"]
99      },
100     {
101         "action": "b",
102         "preconds": ["C"],
103         "add": ["B"],
104         "delete": []
105     }]}
106
107 print(gps(problem2['start'],problem2['finish'],problem2['ops']))
```

Annexe B

Un fragment de géométrie euclidienne en Coq

Nous présentons dans cette annexe un petit fragment de géométrie euclidienne, formalisé directement en Coq. L'intérêt de ce faire était de nous assurer que notre proposition de reconstruction de la géométrie euclidienne était praticable. Nous présentons donc ici la résolution des deux premiers problèmes des *Éléments* (I.1 et I.2) ainsi que les définitions et règles qui sont requises pour cela, et quelques autres tirées du préambule du livre I des *Éléments* ; de nombreuses autres sont cependant omises, comme par exemple tout ce qui concerne les angles.

Nous avouons tout de suite que ce « programme », s'il fonctionne (*i.e.* il est accepté par le vérificateur de preuve Coq), est très grossier, d'abord au niveau du style de Coq, que nous maîtrisons assez mal, mais également au niveau des règles elles-mêmes, dont le nombre peut sans doute être réduit à périmètre égal.

Le programme (reproduit dans le code B.1) est divisé en quatre parties :

1. définitions, règles et axiomes (lignes 3 à 96) ;
2. conventions de notations (lignes 99 à 113) ;
3. règles déduites des définitions et des règles primitives (lignes 114 à 146) ;
4. problèmes I.1 et I.2 (lignes 150 à 207).

Nous commentons seulement la première partie (§ 252) et la dernière (§ 253), de manière assez succincte, sans entrer dans le détail du code, qui requerrait de plus amples développements sur le langage Coq. Afin cependant de donner un aperçu du fonctionnement concret de la vérification de preuve, nous donnons la transcription de l'interaction entre l'utilisateur et Coq pour le début de la démonstration

de l'égalité des longueurs du triangle construit au problème I.1 (§ 254).

§ 252. Définitions, règles et axiomes

Cette partie du programme est elle-même divisée en quatre sous-parties bien visibles :

(a) règles des constructions élémentaires (lignes 3 à 26) ;
(b) règles pour la lecture de points sur des figures (lignes 26 à 47) ;
(c) règles d'algèbre des longueurs (lignes 49 à 81) ;
(d) règles de colinéarité (esquisse) (lignes 85 à 96).

Nous commentons seulement les trois premières sous-parties, la dernière étant une simple esquisse qui n'est pas réutilisée. Chaque règle introduite est suivie d'un petit commentaire dans le code lui-même (encadré par `(* xxx *)` en Coq).

Le premier groupe de règles est le plus important, puisqu'il décrit les actes de *marquage* des points, ainsi que ceux de *tracé* des lignes et des cercles. On peut commencer par ces deux derniers points, qui sont les plus classiques : la l. 19 exprime le postulat 1 (de deux points, tracer une ligne) et la l. 21, le postulat 3 (de deux points tracer un cercle). Les lignes 22 à 25 définissent, pour des raisons de commodité, l'abstraction `figure` sur les points, lignes et cercles. La ligne 26 exprime qu'un polygone est un ensemble de points et de lignes. Il est sous-entendu ici que ces lignes relient ces points ; à strictement parler, on pourrait définir un polygone par la seule suite ordonnée de ses sommets, mais pour des raisons de lourdeur des expressions nous avons directement dédoublé les deux suites.

Le parti-pris le plus original ici (nous semble-t-il) est de considérer que l'acte de marquage d'un point est lui-même significatif et complexe. Mäenpää et Von Plato, par exemple, en font un acte primitif sans condition. Au contraire, nous considérons qu'un point qui est l'intersection de deux lignes n'a pas la même charge d'information qu'un point quelconque du plan. L'idée de la théorie des types étant *de conserver la mémoire des actes réalisés* (leur traçabilité) afin de pouvoir récupérer cette information dans le cours des démonstrations, il faut donc identifier toutes les manières de marquer de points par des règles spécifiques. Ainsi, la première règle (l. 4) demande de marquer un point *distinct* d'une série de figures (par exemple un point distinct de deux autres, ou un point qui n'est pas situé sur un cercle, ni sur une ligne). La seconde règle (l. 6) demande au contraire de marquer un point quelconque sur une figure précise (tout en laissant l'option de préciser qu'il doit

être distinct d'autres figures). Les deux règles suivantes (l. 8 et 10) demandent de prendre l'intersection d'une ligne et d'un cercle ou de deux cercles.

Cette dernière règle peut prêter à discussion, puisqu'il n'est pas sûr qu'il y ait une intersection à deux cercles quelconques ; et lorsque c'est le cas, celle-ci peut être unique (cercles tangents) ou double. Or nous n'associons ici aucune condition à cette règle. Nous laissons ouverte cette question ; d'une part, Euclide lui-même ne la précise pas – c'est un des reproches que lui font les modernes, qui ajoutent en général l'axiome de Pasch à l'axiomatique euclidienne. Or il est aussi possible de considérer qu'une procédure ne porte pas en elle-même ses propres conditions de réussite ; elle peut parfois réussir ou échouer, selon les figures construites initialement. Montrer qu'une procédure est *toujours possible*, par contre, est un acte différent, qui requiert des vérifications et des actes de lecture complémentaires : de la même manière qu'un programme ne dit pas *de lui-même* s'il est correct ou non, mais requiert les méthodes formelles, comme la logique de Hoare, pour ce faire, c'est-à-dire un acte d'observation *au second degré*. Ici, par exemple, la condition la plus simple serait de vérifier que la distance entre les deux centres des cercles est inférieure à la somme de leurs rayons.

Les deux règles suivantes (l. 12 et 14) expriment le postulat 2 d'Euclide, qu'on peut toujours prolonger une ligne à partir de l'une de ses extrémités. Elles correspondent aux deux versions de ce postulat, l'une où on précise la longueur de cette prolongation, l'autre où on la laisse indéterminée. On aurait pu aussi représenter ce postulat par la règle de construction d'une ligne plutôt que d'un point.

Enfin, la règle l. 16 exprime le postulat 5, qu'on peut prolonger deux lignes jusqu'à marquer leur intersection. Là aussi, la condition de non-parallélisme n'est pas exprimée. Un travail complémentaire d'explicitation est requis concernant cette règle.

Nous allons plus vite sur les actes de lectures de points sur des figures (l. 28-47). Par exemple les règles l. 37 et 38 disent que les extrémités d'une ligne sont sur la ligne, celles des lignes 35 et 46 que l'intersection d'une ligne et d'un cercle sont sur cette ligne et ce cercle, etc.

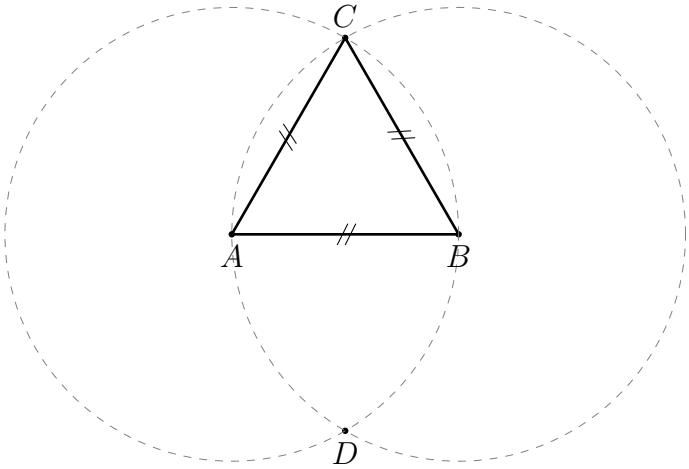
Concernant l'algèbre des longueurs (l. 49-81), on y trouve d'abord la règle dérivée de la définition du cercle (l. 52) qui dit que deux quelconques de ses rayons ont la même longueur. Il y a ensuite les règles qui affirment que l'égalité des longueurs est une relation d'équivalence et enfin des règles qui permettent de la dériver de relations additives. Quant à l'addition des longueurs, elle est définie d'abord par la

propriété fondamentale de la colinéarité ($AB + BC = AC$), puis par sa symétrie et enfin par la substitution possible de l'un de ses termes par la relation d'égalité. On termine cette algèbre (l. 75-81) en définissant la longueur nulle comme celle qui, additionnée à une autre longueur, laisse celle-ci inchangée. Dans le cas où cette ligne est définie par deux points, ces deux points sont égaux.

§ 253. Les problèmes I.1 et I.2

Le problème I.1 des *Éléments* consiste à tracer un triangle équilatéral à partir d'une de ses bases[1].

FIGURE B.1 – Problème I.1 des *Éléments : Triangle équilatéral*



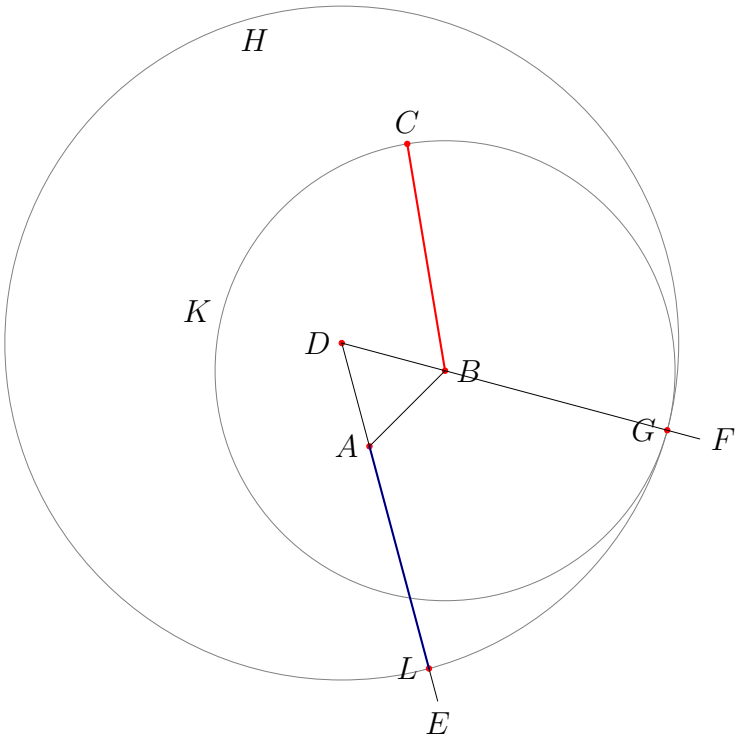
La construction du triangle équilatéral `eqT` à partir de deux points est effectuée dans la définition l. 150-154. La vérification qu'il est équilatéral est l'objet du théorème l. 156-159. Sa preuve (l. 160-169) passe par l'application des règles définies auparavant, introduites par le mot-clé `apply`.

Le problème I.2 consiste à construire, à partir d'un point A donné, un segment d'une longueur égale à celle d'un autre segment BC donné (situé à distance). On sait que le compas d'Euclide est *fermant*, c'est-à-dire qu'il se referme dès qu'il quitte la feuille ; il ne peut reporter les longueurs d'une situation du plan à un autre. Il s'agit donc d'une construction fondamentale, qui affirme l'équivalence du compas fermant et du compas rigide. Elle est admirable à bien des égards par

1. Ici, comme dans nos autres mentions des *Éléments*, nous nous référons à la traduction anglaise commentée et aux diagrammes de (JOYCE 1998) (édition en ligne).

l'économie des moyens dont elle dispose. Nous reproduisons d'abord la démons-

FIGURE B.2 – Problème I.2 des *Éléments* : construction à partir d'un point (A) d'une ligne (AL) égale à une autre (BC)



tration d'Euclide.

Problème. *Placer une droite égale à une droite donnée dont l'une des extrémités se trouve en un point donné.*

Démonstration. Soit A le point donné, et BC la droite donnée.

1. Joindre la droite AB du point A au point B et construire le triangle équilatéral DAB sur cette droite (postulat 1, problème I.1.)
2. Produire les droites AE et BF en ligne droite avec DA et DB (postulat 2).
3. Décrire le cercle CGH de centre B et de rayon BC, et décrire à nouveau le cercle GKL de centre D et de rayon DG (postulat 3).
4. Puisque le point B est le centre du cercle CGH, BC est donc égal à BG. De même, le point D étant le centre du cercle GKL, DL est égal à DG (définition 15).

5. Et dans ces lignes, DA est égal à DB, donc le reste AL est égal au reste BG (notion commune 3).
6. Mais on a aussi prouvé que BC était égal à BG, donc chacune des droites AL et BC est égale à BG. Or, ce qui est égal à la même chose est aussi égal à l'autre, donc AL est aussi égal à BC (notion commune 1).

Par conséquent, la droite AL égale à la droite BC a été placée avec une extrémité au point A. □

La transcription en Coq est assez directe ; elle se divise également en une définition, qui construit la ligne demandée AL (l. 173-180), puis la vérification de l'égalité de sa longueur avec BC, dans un théorème (l. 182-207). La démonstration d'Euclide est suivie par le programme étape par étape. Il faut simplement remarquer que, Coq ne mémorisant pas les noms des constructions intermédiaires (points $D, E, F...$, cercles CGH, GKL, etc.), il faut les réintroduire au moment de la démonstration (l. 184-190) pour la commodité de lecture.

Code B.1 – Fragment de géométrie euclidienne en Coq

```
1  Require Import List.
2
3  Inductive Point : Type :=
4    | Pt : list Figure → Point
5      (* Marquer un point distinct d'un ensemble de figures *)
6    | PtonF : list Figure → Figure → Point
7      (* Marquer un point sur une figure, distinct d'un ensemble d
         'autres figures *)
8    | PtintersLC : Line → Circle → Point
9      (* Marquer l'intersection l'une ligne et d'un cercle *)
10   | PtintersCC : Circle → Circle → Point
11     (* Marquer l'intersection de deux cercles *)
12   | ExtPt1 : Line → Point → Point
13     (*Postulat 2.1: prolonger une ligne, àpartir d'une de ses
         extrémités, jusqu'à un certain point *)
14   | ExtPt2 : Line → Point → Line → Point
15     (*Postulat 2.1: prolonger une ligne, àpartir d'une de ses
         extrémités, d'une certaine longueur, jusqu'à un certain
         point *)
16   | ConvLLPt : Line → Line → Point
```

```
17    (*Postulat 5: Marquer l'intersection de deux lignes (
        requiert que les lignes soient convergentes) *)
18  with Line : Type :=
19   | Ln : Point → Point → Line (* Postulat 1: tracer une ligne entre
        deux points *)
20  with Circle : Type :=
21   | Crc : Point → Point → Circle (*Postulat 3: tracer un cercle à
       partir de deux points *)
22  with Figure : Type :=
23   | P : Point → Figure
24   | L : Line → Figure
25   | C : Circle → Figure.
26  Definition Polygon := prod (list Point) (list Line).
27
28  (* Lecture de points sur des figures *)
29
30  Inductive on : Point → Figure → Prop :=
31   |onF (ff : list Figure) (f : Figure) (H: (In f ff)): on (PtonF ff f) f
32     (* Un point défini comme sur une figure est sur cette figure
          (sauf si déf contradictoire) *)
33   |onLn2 (a b : Point) : on b (L (Ln a (ExtPt1 (Ln a b) b)))
34     (* Le point d'une est sur la ligne qu'on a prolongé àpartir
        de lui *)
35   |onLn3 (ln : Line) (c: Circle) : on (PtintersLC ln c) (L ln)
36     (* L'intersection d'une ligne et d'un cercle est sur la
        ligne *)
37   |onLn4 (a b : Point) : on a (L (Ln a b))
38   |onLn5 (a b : Point) : on b (L (Ln a b))
39     (* Les extremités d'une ligne sont sur la ligne *)
40   |onLn6 (a b c d : Point) (H1: on b (L (Ln a d))) (H2: on c (L (Ln b d))) : on b (L
       (Ln a c))
41   |onC1 (a b : Point) : on b (C (Crc a b))
42     (* Léxtrémité d'un rayon est sur le cercle *)
43   |onC2 (c1 c2 : Circle): on (PtintersCC c1 c2) (C c1)
44   |onC3 (c1 c2 : Circle): on (PtintersCC c1 c2) (C c2)
45     (* L'intersection de deux cercles est sur le cercle *)
46   |onC4 (ln : Line) (c: Circle) : on (PtintersLC ln c) (C c).
47     (* L'intersection d'une ligne et d'un cercle est sur le
```

```
      cercle *)
48
49 (* Algebre des longueurs *)
50
51 Inductive eqL: Line → Line → Prop :=
52   | sameCrc (a b c: Point) (H: on b (C (Crc a c))) : eqL (Ln a c) (Ln a b)
53     (* Si B est sur le cercle (A,AC) alors AB = AC *)
54   | reflEqPt (a b : Point) : eqL (Ln a b) (Ln b a) (* AB = BA *)
55   | reflEq (ln : Line) : eqL ln ln (* L = L *)
56   | symEq (ln1 : Line) (ln2 : Line) (H1: eqL ln1 ln2): eqL ln2 ln1
57     (* Si L1 = L2 alors L2 = L1 *)
58   | transEq (ln1 : Line) (ln2 : Line) (ln3 : Line)(H1: eqL ln1 ln2) (H2 : eqL
       ln2 ln3) : eqL ln1 ln3
59     (* Si L1 = L2, L2 = L3, L1 = L3 *)
60   | substadd1 (ln1 ln2 ln3 ln4: Line) (H1: addL ln1 ln2 ln3)(H2: addL ln1
       ln2 ln4): eqL ln3 ln4
61     (*Si L3 = L1 + L2, et L4 = L1 + L2, alors L3 = L4 *)
62   | substadd2 (ln1 ln2 ln3 ln4: Line) (H1: addL ln1 ln2 ln3)(H2: addL ln4
       ln2 ln3): eqL ln1 ln4
63     (*Si L3 = L1 + L2, et L3 = L4 + L2, alors L1 = L4 *)
64 with addL: Line → Line → Line → Prop :=
65   | coladd (a b c: Point) (H: on b (L (Ln a c))): addL (Ln a b) (Ln b c) (Ln a c)
66     (* Si A,B,C sont alignés, alors AB + BC = AC *)
67   | symadd (ln1 ln2 ln3: Line) (H1: addL ln1 ln2 ln3) : addL ln2 ln1 ln3
68     (* L1 + L2 = L2 + L1 *)
69   | substadd3 (ln1 ln2 ln3 ln4: Line) (H1: addL ln1 ln2 ln3)(H2: eqL ln1 ln4
       ): addL ln4 ln2 ln3
70     (* Si L1 = L4, alors L1 + L2 = L4 + L2 *)
71   | substadd4 (ln1 ln2 ln3 ln4: Line) (H1: addL ln1 ln2 ln3)(H2: eqL ln3 ln4
       ): addL ln1 ln2 ln4.
72     (* Si L1 + L2 = L3 et L3 = L4, alors L1 + L2 = L4 *)
73 Definition minusL (ln1 ln2 ln3: Line) : Prop := addL ln3 ln2 ln1.
74
75 (* Identité de deux points si leur longueur est nulle *)
76
77 Inductive eqPt : Point → Point → Prop :=
78   | eqPt0 (a b : Point) (ln : Line) (H: addL (Ln a b) ln ln): eqPt a b.
79     (* Si une longueur est nulle entre deux points, ils
```

```
      coincident *)

Axiom identityPt: forall (a b: Point), eqPt a b → a = b.

(* Colinéarité *)

Inductive colinear : Point → Point → Point → Prop :=
 |colsym (a b c : Point) (H: colinear c a b): colinear a b c
   (* Si A,B,C sont alignés, alors C,A,B aussi *)
 |colon (a b c : Point) (H: on c (L (Ln a b))): colinear a c b
   (* Si C est sur AB, alors A,B,C sont alignés *)
 |colconv1 (a b : Point) (ln: Line) : colinear a b (ConvLLPt (Ln a b) ln)
 |colconv2 (a b : Point) (ln: Line) : colinear a b (ConvLLPt ln (Ln a b)).
   (* Deux points sont colinéaires avec l'intersection de leur
      ligne et d'une autre *)

Axiom colin: forall (a b c: Point), colinear a b c ↔ (on c (L (Ln a b)) ∨ on b
    (L (Ln a c)) ∨ on a (L (Ln b c))).
   (* Axiome de colinéarité: Si A,B,C sont alignés, alors l'un
      des trois points est entre les deux autres *)

(* Notations *)

Notation "x :: l" := (cons x l)
                  (at level 60, right associativity).
Notation "[ ]" := nil.
Notation "[ x ; .. ; y ]" := (cons x .. (cons y nil) ..).
   (* Notations usuelles des listes *)
Notation "p !P! n " := (nth n (fst p) (Pt nil))
                   (at level 90).
   (* Désigne le n-ième point du polygone *)
Notation "p !L! n " := (nth n (snd p) (Ln (Pt nil) (Pt nil)))
                   (at level 90).
   (* Désigne la n-ième ligne du polygone *)
Notation "l1 == l2" := (eqL l1 l2)
                  (at level 60, right associativity).
Notation "l1 ++l2 == l3" := (addL l1 l2 l3)
                  (at level 70, right associativity).
```

```
114     (* Egalités et additions des longueurs *)
115
116  (* Quelques petits théorèmes utiles *)
117
118  Theorem substadd2' : forall (ln1 ln2 ln3 ln4 : Line), (addL ln1 ln2 ln3) ∧
        (addL ln1 ln4 ln3) → eqL ln2 ln4.
119     (* Si L1 + L2 = L1 + L4, alors L2 = L4 *)
120  Proof. intros ln1 ln2 ln3 ln4 H. destruct H as [H1 H2].
121    apply symadd in H1. apply symadd in H2. apply substadd2 with (ln2 :=
         ln1) (ln3 := ln3). apply H1. apply H2. Qed.
122
123  Theorem symEqPt: forall (a b : Point) (ln : Line), ((Ln a b) == ln) → ((Ln b a)
        == ln).
124     (* Si (AB) = L, alors (BA) = L *)
125  Proof. intros a b ln H.
126  apply transEq with (ln2 := Ln a b). apply reflEqPt. apply H. Qed.
127
128  Theorem symEqPt' : forall (a b : Point) (ln : Line), (ln == (Ln a b)) → (ln == (
        Ln b a)).
129     (* Si L = (AB) = L, alors L = (BA) *)
130  Proof. intros a b ln H.
131  apply transEq with (ln2 := Ln a b). apply H. apply reflEqPt. Qed.
132
133  Theorem onCrc: forall (a b c: Point), (Ln a b == Ln a c) → on b (C (Crc a c)).
134     (* Si AB = AC, alors B est sur le cercle (A,AC)*)
135  Proof. intros a b c H.
136  remember (PtintersLC (Ln a b) (Crc a c)) as d.
137  assert (H1: d = b).
138    {apply identityPt. apply eqPt0 with (ln:= Ln a b).
139     apply symadd. apply substadd3 with (ln1 := Ln a d).
140     - apply coladd. rewrite Heqd. apply onLn3.
141     - apply symEq. apply transEq with (ln2 := Ln a c).
142       + assumption.
143       + apply sameCrc. rewrite Heqd. constructor. }
144  rewrite ← H1.
145  rewrite Heqd. constructor.
146  Qed.
147
```

```
148 (* Problème I.1*)
149
150 Definition eqT (a b : Point) : Polygon :=
151 let (c1,c2) := (Crc a b,Crc b a) in
152 let c := PtintersCC c1 c2 in
153     ([ a ;  b;  c],[  (Ln a b);   (Ln a c);   (Ln b c)]).
154 Check eqT.
155
156 Theorem equT: forall (a b : Point),
157     (( eqT a b) !L! 0 )  == ((eqT a b) !L! 1 ) ∧
158     (( eqT a b) !L! 0 )  == ((eqT a b) !L! 2 ) ∧
159     (( eqT a b) !L! 1 )  == ((eqT a b) !L! 2 ).
160 Proof. intros a b.
161   assert (((eqT a b) !L! 0 ) == ((eqT a b) !L! 1 )).
162     { unfold eqT. simpl. apply sameCrc. apply onC2. }
163   assert (((eqT a b) !L! 0 )  == ((eqT a b) !L! 2 )).
164     {   unfold eqT. simpl. apply symEqPt. apply sameCrc. apply onC3. }
165   split. assumption. split. assumption.
166   apply transEq with (ln2 := (eqT a b !L! 0 )).
167   - apply symEq. assumption.
168   - apply H0.
169 Qed.
170
171 (*Problème I.2 : construire un segment de longueur BC àpartir
       du point A*)
172
173 Definition constrI2 (a b c : Point) : Line :=
174 let d := (eqT a b) !P! 2  in
175 let e := ExtPt1 (Ln d a) a in
176 let f := ExtPt1 (Ln d b) b in
177 let cgh := Crc b c in
178 let g := PtintersLC (Ln b f) cgh in
179 let gkl := Crc d g in
180 let l := PtintersLC (Ln a e) gkl in Ln a l.
181
182 Theorem checkI2 : forall (a b c : Point), (constrI2 a b c) == (Ln b c).
183 Proof. intros a b c. unfold constrI2.
184 remember ((eqT a b) !P! 2 ) as d.
```

```
185 remember (ExtPt1 (Ln d a) a) as e.
186 remember (ExtPt1 (Ln d b) b) as f.
187 remember (Crc b c) as cgh.
188 remember (PtintersLC (Ln b f) cgh) as g.
189 remember (Crc d g) as gkl.
190 remember (PtintersLC (Ln a e) gkl) as l.
191 apply transEq with (ln2 := (Ln b g)).
192 - apply substadd2' with (ln1 := (Ln d a)) (ln3 := (Ln d l)).
193   split.
194   + apply coladd. apply onLn6 with (d := e).
195     * subst e. apply onLn2.
196     * subst l. apply onLn3.
197   + apply substadd4 with (ln3 := (Ln d g)).
198     apply substadd3 with (ln1 := (Ln d b)).
199     * apply coladd. apply onLn6 with (d := f).
200       subst f. apply onLn2. subst g. apply onLn3.
201     * assert (H: (Ln a b) == (Ln a d) ∧ (Ln a b) == (Ln b d) ∧ (Ln a d) == (Ln b d))
          .
202         { subst d. apply (equT a b). }
203       destruct H as [H1 [H2 H3]].
204       apply symEqPt. apply symEqPt'. apply symEq. assumption.
205     * apply sameCrc. subst l. subst gkl. apply onC4.
206 - apply symEq. apply sameCrc. subst g. subst cgh. apply onC4.
207 Qed.
```

§ 254. Illustration de la vérification de preuve

Afin d'illustrer le fonctionnement de Coq, nous donnons l'interaction détaillant une partie de la démonstration du théorème `equT`, qui affirme l'égalité du premier côté construit du triangle avec sa base. Dans le code B.1, il s'agit de la ligne 161 (énoncé de la proposition) et 162 (preuve).

Cette interaction est reproduite dans le code B.2. Le fragment du théorème est énoncé en ligne 1, après l'invite de commande >>> : il dit que pour tout segment `ab`, le triangle équilatéral `eqT a b` construit sur cette base a sa première ligne (`!L! 0`, selon notre convention d'écriture, *i.e.*`ab`) égale à sa deuxième (`!L! 1`, selon notre convention `ac`, où `c` est le sommet construit de `eqT a b`). La preuve comporte quatres instructions données à la machine, qui peuvent être localisées

dans les lignes 5,10, 15, 20, introduites par l'invite de commande >>> et qui sont affichées en gras.

CODE B.2 – Preuve interactive d'un fragment de la démonstration I.1

```
1  >>>  Theorem equT0: forall (a b : Point),
   ((eqT a b) !L! 0) == ((eqT a b) !L! 1).
2  1 goal
3    ============================
4    forall a b : Point, (eqT a b !L! 0) == (eqT a b !L! 1)
5  >>> intros a b.
6  1 goal
7    a, b : Point
8    ============================
9    (eqT a b !L! 0) == (eqT a b !L! 1)
10 >>> simpl.
11 1 goal
12   a, b : Point
13   ============================
14   Ln a b == Ln a (PtintersCC (Crc a b) (Crc b a))
15 >>> apply sameCrc.
16 1 goal
17   a, b : Point
18   ============================
19   on (PtintersCC (Crc a b) (Crc b a)) (C (Crc a b))
20 >>> apply onC2.
21 No more goals.
22 >>> Qed.
```

Les paragraphes qui suivent chacune de ces instructions sont les réponses affichées par l'assistant de preuve. Chaque paragraphe a la même structure, qui décrit le contexte de la preuve, d'abord le nombre d'objectifs à atteindre (`goal`), par exemple ligne 11. Le contexte lui-même est divisé en deux parties par le délimiteur ==========, par exemple ligne 13. Au-dessus de ce délimiteur se trouvent les données (objets et hypothèses disponibles). La ligne 12 par exemple indique qu'on dispose de deux points avec lesquels on peut raisonner. Au-dessous du délimiteur se trouve la proposition à démontrer, par exemple ligne 14, que le segment `ab` est égal au segment `ac`, où le point `c` apparaît directement dans sa définition constructive, comme intersection des deux cercles de rayon `ab` et de centres `a` et `b` respectivement. Lorsqu'on propose de résoudre cet objectif (ligne 15) par l'application de la règle `sameCrC`, qui affirme que le segment reliant le centre d'un cercle

à l'un de ses point quelconques est égal à son rayon (code B.1, l. 52), cet objectif se transforme et requiert l'investigateur de montrer à présent que le point `c` est bien sur le cercle de centre `a` et de rayon `ab` (ligne 19). Cela se fait (ligne 20) en appliquant la règle `onC2`, qui dit que l'intersection de deux cercles est sur le premier cercle(code B.1, l. 43). Au début de la preuve, l'instruction 5 sert à éliminer le quantificateur universel `forall` et l'instruction 10 `simpl` demande à Coq de simplifier l'expression en la réduisant aux constructeurs élémentaires de la théorie et aux hypothèses `a,b`.

Mais pour l'investigateur qui est en train de construire la théorie elle-même, comme nous le fûmes dans cette tentative de formalisation de la géométrie euclidienne, ce *feedback* interactif que donne Coq permet de voir immédiatement quelles règles sont encore manquantes dans la théorie. Dans ce cas particulier, il pointe vers la nécessité d'une règle aussi élémentaire que celle affirmant que le point intersectant deux figures (lignes ou cercles) est bien sur chacune d'entre elle, qui se décline en plusieurs cas de figure (dont la règle `onC2`).

Il faut remarquer que la nécessité de ces règles provient des choix de conception réalisés, et qu'elles pourraient sans doute être reformulées ou rendues inutiles par d'autres choix. L'assistant de preuve permet donc également de réfléchir à la structure de la théorie. Dans notre illustration, un problème ouvert est par exemple de savoir comment, dans les choix théoriques réalisés, formuler la condition d'existence de l'intersection entre deux cercles, qui est admise par Euclide, et qui est l'objet de nombreuses propositions depuis les Temps modernes (l'axiome de Pasch étant la plus célèbre aujourd'hui)[2]. Aussi l'assistant de preuve est-il un espace de travail de l'élaboration théorique, qui permet d'inspecter quelles règles pourraient être factorisées parce que d'usage similaire, quels schémas d'induction sont requis pour donner au système sa forme générative, quelles définitions devraient être réécrites pour être de manipulation plus aisée, etc. Coq n'aide donc pas seulement à écrire des preuves systématiques, mais à explorer et à construire le système lui-même.

2. (De Risi 2020).

Table des extraits de code

Table des figures

Glossaire

Les définitions techniques présentées ici le sont uniquement pour la convenance de la lectrice. Elles ne visent pas l'exactitude et l'élaboration, mais le repérage rapide du sens des termes de manière à ne pas rompre le fil de la lecture.

λ-Calcul Système formel conçu par Alonzo Church dans les années 1930 qui permet de décrire la notion de calcul et de manipulation de fonctions de manière abstraite. Les expressions du λ-calcul dénotent des fonctions, que des règles permettent de composer, d'abstraire et d'appliquer. p. 166, 590, 632, 910

π-Calcul Système formel conçu par Robin Milner dans les années 1990 pour décrire et analyser le comportement des systèmes de calculs parallèles et distribués. p. 240, 910

Actuateur Composant qui transforme un signal d'entrée (électrique ou autre) en une action physique, par exemple ouvrir une vanne. p. 236, 882

Administration système Gestion et maintenance des systèmes informatiques, des réseaux et des serveurs, y compris les tâches de configuration, de surveillance, de dépannage et de sécurité. p. 96, 176

Affectation de ressources (*provisioning*). Processus d'allocation et de gestion automatisée des ressources informatiques, telles que les serveurs, les réseaux et le stockage, pour répondre aux besoins des applications et des services. p. 909

Agile Approche de développement d'applications axée sur la livraison fréquente de fonctionnalités, la collaboration, l'implication du maître d'ouvrage, et l'adaptation continue. L'Agilité vise à fournir des solutions de haute qualité de manière flexible et réactive. p. 125, 127, 131, 133, 134, 136, 138, 139, 165, 197, 200, 201, 217, 226, 227, 909, 937, 941

Algorithme bayésien Algorithme basé sur le théorème de Bayes, utilisé pour es-

timer la probabilité d'un événement en utilisant des probabilités conditionnelles. p. 935

Algorithme de Gale-Shapley Méthode de résolution de problèmes d'appariement qui assure, en fonction des préférences des individus, que chacun soit associé à un partenaire de telle manière qu'aucune paire d'individus non appariés ne souhaite rompre leur association existante pour former un nouveau couple. p. 970

Algorithme génétique Méthode d'optimisation inspirée de la sélection naturelle, utilisant des techniques de reproduction, de mutation et de croisement pour évoluer vers des solutions de plus en plus adaptées à un problème donné. p. 935

Algorithmes incrémentaux (*online algorithms*). Algorithmes qui effectuent leur tâche au fur et à mesure de la réception des données, plutôt que sur des données définitives. p. 33

API (*Application Programming Interface*). Ensemble de règles et de protocoles permettant à différentes applications de communiquer entre elles. p. 129

Apprentissage machine Approche de définition ou d'amélioration de fonctions informatiques basée sur l'exploitation cumulative de données, sans programmation explicite. p. 83, 93, 142, 172, 275, 278, 286

Apprentissage profond Approche de l'apprentissage machine qui utilise des réseaux de neurones artificiels avec plusieurs couches pour modéliser et résoudre des problèmes complexes. p. 20, 93, 629, 858, 934

Architecture logicielle Structure globale d'un logiciel qui détermine la manière dont les composants et les modules interagissent, facilitant la conception, la maintenance et l'évolution du logiciel. p. 114

ASP (*Application Service Provider*). Fournisseur de services applicatifs. Entreprise fournissant des applications en tant que services via Internet. p. 98

Assembleur Langage de programmation proche du langage machine, utilisant des codes mnémoniques pour représenter les instructions exécutables directement par le processeur de la machine cible. p. 159

Automate fini Modèle mathématique de calcul représentant une machine à états avec un nombre fini d'états et des transitions entre ces états. Les automates finis sont utilisés pour reconnaître des langages réguliers et des motifs simples. p. 177

Automatisation robotisée des processus *RPA* (*robotic process automation*). Tech-

nologie d'automatisation de tâches répétitives, souvent liées à des interactions à des manipulations de données ou d'applications au sein de flux de tâches*. p. 901

Back-end La partie d'une application logicielle qui gère les traitements en arrière-plan, telles que les interactions avec la base de données et la logique métier. p. 128–131

BIM (*Building Information Modeling*). Ensemble de techniques et de standards de modélisation et de gestion numériques des informations liées à la construction. Elles permettent de créer des représentations et des simulations diverses à propos d'une infrastructure (luminosité, efficacité énergétique, etc.). p. 849, 854, 914

Bit Plus petite unité de données dans un système informatique, pouvant prendre la valeur 0 ou 1. p. 174

Blockchain Technologie de registre distribué utilisée pour sécuriser et valider les transactions en enregistrant des données sous forme de blocs interconnectés et sécurisés cryptographiquement. p. 72, 74, 898

Bogue (*bug*). Erreur ou défaut dans un programme qui provoque un comportement inattendu ou incorrect. p. 856

Booléen Type de données pouvant prendre deux valeurs, généralement vrai (*true*) ou faux (*false*). p. 224

CAO Conception assistée par ordinateur (anglais : *CAD, Computer aided design*). Méthode utilisant des logiciels pour faciliter la création, la modification et l'optimisation de modèles et de dessins techniques. p. 101, 277, 808, 848, 849, 854

Capteur (*sensor*). Dispositif électronique qui mesure une grandeur physique, telle que la température, la lumière ou la pression, et la convertit en signal électrique. p. 236, 882, 883, 885

Classe Concept fondamental de la programmation orientée objet, définissant les propriétés et les comportements d'objets (ou encore leurs données et leurs procédures). p. 907

Compilé Programme, initialement écrit dans un langage de haut niveau, transformé en langage machine. Le programme effectuant cette transformation est appelé compilateur. . p. 6, 170, 173, 184, 189, 198, 209, 271

Connexionnisme Approche en intelligence artificielle basée sur le modèle du cer-

veau humain, où l'apprentissage est simulé par des connexions entre unités de traitement. p. 286

Conteneurisation Méthode de déploiement de logiciels dans des environnements autonomes incluant leurs dépendances et configurations. Cela permet aux applications d'être exécutées de manière isolée et cohérente, indépendamment de l'infrastructure sous-jacente, facilitant ainsi leur déploiement et leur maintenance. p. 125

Contrat intelligent Programme informatique qui exécute automatiquement les termes d'un contrat lorsque ses conditions sont remplies, généralement basé sur la technologie de la blockchain. p. 969

Dette informatique (*Legacy*) Expression désignant la difficulté et le coût d'évolution ou de remplacement d'un système informatique complexe, lié à l'obsolescence de ses technologies sous-jacentes ou à la perte de connaissance de son fonctionnement interne. Le remplacement peut être complexe ou dangereux car le système effectue des tâches essentielles dont on a cependant perdu la spécification. p. 133, 1009

Devops Approche qui vise à intégrer le développement et l'exploitation d'applications afin d'améliorer la qualité, l'efficacité et la vitesse d'évolution en automatisant les processus et en favorisant la collaboration entre les équipes. p. 80, 126, 201, 909

Diagramme de flux Diagramme visuel utilisé pour représenter la séquence des étapes d'un processus ou d'un algorithme, montrant la direction des données ou du contrôle entre ces étapes. p. 156, 157, 189, 781

Dictionnaire Structure de données associant des clés à des valeurs, permettant un accès efficace et rapide aux éléments en utilisant les clés. p. 764

Distribué Système ou réseau informatique où les ressources et les traitements sont répartis sur plusieurs machines, ce qui permet de répartir la charge et d'améliorer la résilience. p. 207

Donnée annotée Donnée à laquelle des métadonnées ou des informations contextuelles ont été ajoutées pour fournir des informations supplémentaires sur sa signification et son utilisation. p. 232

Débogué Code ou programme dont les erreurs et les problèmes ont été identifiés et corrigés. p. 119, 189

EDI *Electronic Data Interchange*. Échange électronique de données. Processus

automatisé d'échange d'informations transactionnelles entre systèmes informatiques, favorisant l'efficacité et la précision dans les transactions. p. 197, 211, 896, 902

ERP *Enterprise Resource Planning*. Système intégré de gestion qui gère les opérations, les processus et les ressources d'une organisation pour améliorer l'efficacité et la prise de décision. Il inclut en général le système informatique comptable, et souvent le système logistique, parfois le système commercial ou le système de ressources humaines. p. 197, 202, 229, 230, 351, 889, 893, 896, 974

Exception (dans un programme) Situation anormale ou inattendue qui interrompt le flux normal d'exécution et peut nécessiter une gestion spécifique. p. 131

Flux de tâches (*workflow*) Séquence ordonnée d'activités ou d'étapes d'un processus, faisant intervenir en général plusieurs agents (automates ou personnes). p. 151, 354, 1007

Forêt aléatoire (*random forest*). Algorithme d'apprentissage machine statistique modélisant un ensemble d'arbres de décision entraînés sur des sous-ensembles aléatoires de données, utilisés pour la classification et la prédiction. p. 935

Front-end La partie d'une application logicielle avec laquelle l'utilisateur interagit directement, souvent responsable de l'interface utilisateur et de la collecte des données. p. 128, 131

GPX format standard de présentation de coordonnées GPS constituant un itinéraire, utilisé pour des activités de plein air et pour la navigation. p. 353

Grand modèle de langage (*large language model*). Modèle d'intelligence artificielle capable de comprendre et de générer du langage humain de manière avancée, en utilisant des techniques d'apprentissage profond. p. 935, 941, 942, 945, 948

Greenfield Projet de développement d'application démarrant à neuf, sans contraintes liées à la dette informatique*. p. 118

Hacker Individu ayant des compétences en informatique et en sécurité, souvent impliqué dans la découverte de vulnérabilités, la résolution de problèmes complexes et le développement de solutions innovantes. p. 134, 966

Informatique en nuages Modèle de prestation de services informatiques via Internet, permettant un accès flexible et à la demande à des ressources telles que

des serveurs, des bases de données et des applications. p. 79, 96, 97, 125, 860, 874, 888, 896

Intégration continue Pratique de développement logiciel où les modifications de code sont régulièrement intégrées dans une branche principale, ce qui permet de détecter rapidement les problèmes d'intégration et de garantir la stabilité du code. p. 125, 909

Invite de commande (*command prompt*). Interface textuelle permettant à l'utilisateur d'entrer des commandes pour interagir avec un système d'exploitation ou un programme. p. 108, 142

Jeu d'arcade Type de jeu vidéo caractérisé par des actions rapides, des graphismes simples et une jouabilité immédiate. p. 151

Langage-machine Langage de programmation bas niveau constitué d'instructions directement compréhensibles par un processeur, utilisé pour écrire des programmes exécutables. p. 184

Liaison tardive (*late binding*) Processus de liaison des fonctions ou méthodes d'un programme au moment de l'exécution plutôt qu'à la compilation. p. 783

Librairie Ensemble de fonctions, de modules ou de composants réutilisables pour le développement de logiciels. p. 20, 173, 209, 229, 271

Liste Structure de données linéaire contenant une séquence ordonnée d'éléments, avec un accès facile au début ou à la fin. p. 764, 770

Log Enregistrement séquentiel d'événements ou d'actions dans un système informatique, utilisé pour le dépannage, le suivi et l'analyse. p. 353

Machine virtuelle Environnement logiciel isolé et autonome qui émule une machine physique et permet l'exécution de programmes indépendamment du système hôte. p. 40, 874

Mainframe Ordinateur puissant utilisé pour exécuter des applications critiques et gérer de gros volumes de données. p. 165

Masquage des données (*data hiding*). Technique de programmation qui consiste à restreindre l'accès direct aux données internes d'un objet, afin d'assurer l'encapsulation et la protection des informations sensibles. p. 199

Micro-contrôleur Circuit intégré autonome contenant un processeur, une mémoire et des interfaces, utilisé pour contrôler des systèmes embarqués et des appareils électroniques. p. 811

Micro-services Architecture logicielle où une application est décomposée en services indépendants, chacun exécutant une fonction spécifique et interagissant via des interfaces définies. p. 125, 138

Middleware Logiciel intermédiaire qui facilite la communication et la gestion des données entre différentes applications ou composants logiciels. p. 231

Mise en cache (*caching*). Techniques d'optimisation de transfert de données, qu'on anticipe lors de périodes de disponibilité du canal de communication, et qu'on enregistre en mémoire locale jusqu'à leur consommation. p. 33

Mise en production Processus de déploiement d'un logiciel ou d'un système dans un environnement de production opérationnel, rendant les nouvelles fonctionnalités accessibles aux utilisateurs finaux. p. 125, 201, 241, 907, 909

Modèle de développement en V Approche de développement logiciel où chaque étape de programmation est associée à une phase de validation ou de vérification correspondante, formant ainsi une structure en forme de V. p. 123

Modèle en cascade Méthodologie de développement logiciel linéaire où les phases de conception, de développement, de test et de déploiement sont effectuées séquentiellement, en suivant un flux descendant. p. 123, 133, 186

Monte Carlo Approches statistiques utilisant des échantillons aléatoires pour estimer des résultats numériques. Elles sont utilisées en simulation, en analyse de risque et en modélisation numérique. p. 936

Mémoire vive Mémoire volatile utilisée par un ordinateur pour stocker temporairement les données et les programmes en cours d'utilisation. Les données sont rapidement accessibles mais sont effacées lorsque l'ordinateur est éteint. p. 184

Méthode formelle Approche de développement logiciel basée sur des techniques mathématiques et des spécifications formelles pour garantir la fiabilité et la vérification des systèmes. p. 82, 184, 186, 197, 228, 844

Objet Unité fondamentale en programmation orientée objet. Un objet est une instance d'une classe possédant des propriétés et des méthodes. p. 130

Open-source Modèle de développement de logiciel où le code source est librement accessible, modifiable et distribuable, encourageant la collaboration et la transparence. p. 91, 97–99, 101, 102, 966

Opérations (informatiques) Ensemble des tâches et des activités liées à la gestion, à la supervision et à l'entretien des systèmes informatiques en production. p. 96, 99, 201, 907, 909

Oracle En informatique théorique, une source d'information externe qui peut fournir des réponses instantanées (sans coût computationnel) à des questions spécifiques. Il est utilisé pour étudier la complexité des algorithmes. p. 754

Ordinateur analogique Dispositif de calcul qui manipule des quantités continues plutôt que des valeurs discrètes, utilisé pour des calculs impliquant des grandeurs physiques. p. 142

Ordinateur de Von Neumann Modèle d'ordinateur où instructions et données sont stockées dans la même mémoire principale. Une unité de contrôle identifie séquentiellement les instructions à traiter et les soumet à l'unité de traitement. Mémoire, unité de contrôle et unité de traitement sont les trois composants centraux de ce modèle qui est à la base de la plupart des ordinateurs modernes. p. 924

Paquetage (*package*). Ensemble organisé de logiciels, de librairies, de fichiers de configuration et de « métadonnées », utilisé pour faciliter l'installation, la gestion et la distribution d'application sur un système d'exploitation. p. 911

Paradigme de programmation Ensemble cohérent de concepts et de méthodes qui guident l'approche de conception et d'écriture de programmes informatiques. p. 81, 141, 277, 910

Parallélisme Exécution simultanée de plusieurs processus, en général en interaction ou interdépendants. p. 32

Pile (*Stack*). Structure de données de type LIFO (Last-In-First-Out), c'est-à-dire où les données peuvent être accédées dans l'ordre inverse de leur mémorisation. p. 174, 770, 781

Pile technologique (*technology stack*). Combinaison de technologies, de langages de programmation, de modèles et d'outils utilisés pour développer une application ou un projet informatique. p. 125, 126, 133

Porte quantique Opération unitaire sur un qubit ou un ensemble de qubits dans un ordinateur quantique, utilisée pour effectuer des transformations sur les états quantiques. p. 835

Problème de satisfiabilité booléenne (SAT) Problème de déterminer s'il existe une assignation de valeurs de vérité aux variables d'une expression booléenne telle que l'expression entière soit évaluée comme vraie. p. 736, 769

Problème du cycle hamiltonien Problème consistant à trouver un cycle passant par chaque nœud d'un graphe, sans passer deux fois par le même nœud.

p. 769, 1013

Problème du voyageur de commerce ou TSP (*Traveling Salesman Problem*). Il s'agit d'une sophistication du problème du cycle hamiltonien*, où chaque arête entre deux nœuds a une distance ; il s'agit alors de trouver le cycle hamiltonien le plus court. Dans sa formulation traditionnelle, il s'agit de trouver le chemin le plus court qui permet à un voyageur de visiter toutes les villes d'une région, chacune une fois seulement, et de revenir à son point de départ. p. 775

Programmation concurrente Technique de programmation visant l'exécution de plusieurs tâches simultanées, voire en interaction, afin de résoudre des problèmes complexes plus efficacement. p. 141

Programmation fonctionnelle Paradigme de programmation basé sur l'utilisation de fonctions comme éléments de base, évitant de modifier l'état des ressources mobilisées, contrairement à la programmation impérative. p. 141, 161, 165, 237, 277, 916

Programmation impérative Paradigme de programmation où les instructions sont données dans l'ordre séquentiel à suivre par le programme, définissant explicitement les étapes à exécuter. p. 141, 151, 622, 845

Programmation logique Paradigme de programmation basé sur la logique formelle, où les programmes sont exprimés en termes de relations logiques et de règles d'inférence. p. 144, 277, 314, 845

Programmation objet Paradigme de programmation basé sur l'organisation du code en objets interagissant par envoi de messages, favorisant l'encapsulation et la réutilisation du code. p. 124, 134, 141, 620, 890

Programmation par contraintes Paradigme de programmation où les problèmes sont formulés en termes de contraintes et de relations entre les variables, permettant de résoudre des problèmes d'optimisation et de satisfaction de contraintes. p. 845

Programmation structurée Méthode de programmation qui encourage l'utilisation de structures de contrôle telles que les boucles et les conditions pour rendre le code plus lisible, compréhensible et modulaire. p. 123, 134, 135, 199

Qubit Unité de base de l'information quantique, correspondant à une superposition des états 0 et 1, utilisée dans les ordinateurs quantiques. p. 835, 837

Queue Structure de données de type FIFO (First-In-First-Out) utilisée pour sto-

cker et gérer des données. p. 770

Randomisation Méthode statistique visant à introduire de l'aléatoire dans la conception d'expériences ou d'études pour minimiser les biais et les effets indésirables. p. 32

Redondance Duplication d'informations, de composants ou de systèmes dans le but d'améliorer la fiabilité, la tolérance aux pannes ou la disponibilité. p. 32

Registre Petite zone de mémoire rapide située dans un processeur, utilisée pour stocker temporairement des données et des instructions pendant l'exécution d'un programme. Également, une structure de donnée utilisée pour stocker des informations persistantes. C'est ce second sens que nous généralisons dans notre dernier chapitre. p. 159

Retour sur trace (*backtracking*). Méthode de recherche de solutions qui explore des arbres de possibilité en profondeur puis revient sur ses pas lorsqu'elle rencontre une impasse. p. 33

Règle métier (*business rule*). Directive décrivant un processus, une contrainte ou une logique spécifique à une organisation, utilisée pour guider les décisions et les opérations. p. 118, 203

Réseau de neurones Modèle d'apprentissage machine inspiré du fonctionnement des neurones biologiques, utilisé pour résoudre des tâches complexes de classification et de prédiction. p. 629, 935

Réusinage (*refactoring*). Processus d'amélioration de la structure interne du code source sans en modifier le comportement externe, visant à améliorer la maintenabilité et la lisibilité. p. 126

SaaS (*Software as a Service*). Application fournie comme un service, généralement via Internet et un navigateur Web. p. 98

Script Petit programme ou ensemble d'instructions exécutées dans le cadre de l'usage d'une application ou d'un système informatique, servant en général à automatiser des tâches répétitives ou manipuler des données. p. 96, 100, 153, 154, 168, 172, 178

Scrum Méthode de gestion de projet agile basée sur des itérations courtes appelées sprints, avec une approche collaborative et itérative. p. 128

SOA (*Service Oriented Architecture*). Approche de conception logicielle où les fonctionnalités sont décomposées en services indépendants et interopérables. p. 125, 228

Sprint Période de temps fixe dans la méthodologie agile Scrum, généralement de 1 à 4 semaines, pendant laquelle une équipe de développement travaille sur un ensemble de tâches et de fonctionnalités définies. p. 125, 200

Système cyber-physique Système intégrant des composants physiques et des systèmes informatiques interconnectés. Les CPS interagissent avec le monde réel en utilisant des capteurs et des actuateurs pour des applications diverses. p. 242, 350, 873

Système d'exploitation Logiciel de base qui gère les ressources matérielles et permet aux autres logiciels (applications) de s'exécuter sur un ordinateur. p. 6, 97, 125, 231, 879

Système d'information Ensemble de ressources, de méthodes et de technologies qui collectent, stockent, traitent et diffusent des informations pour soutenir les opérations d'une organisation. p. 79, 966

Système de production Système de calcul proposé par Emil Post dans les années 1920 qui permet de définir des ensembles de règles pour la transformation itérative de chaînes de symboles selon des motifs spécifiés. p. 166, 270, 310, 314

Système embarqué Système informatique intégré dans un appareil ou un système plus vaste, conçu pour effectuer des tâches spécifiques et souvent dédié à une application particulière. p. 239

Sémaphore Mécanisme de synchronisation utilisé pour contrôler l'accès concurrentiel à des ressources partagées dans les systèmes informatiques. p. 120

Sûreté (*safety*) Propriété d'un programme qu'il ne rencontrera *jamais* une situation (ou un état) indésirable au cours de son exécution. p. 32, 844

Tableur Programme permettant d'organiser des calculs numériques de manière chaînée, comme Microsoft Excel. p. 73, 90, 94, 148, 178, 765, 920

Test en masse Processus de vérification et de validation d'une application ou d'un système en soumettant des charges de travail et des scénarios variés pour évaluer ses performances et sa robustesse. p. 214

Thèse de Church-Turing Principe affirmant que tout problème calculable peut être résolu par une machine de Turing, établissant ainsi la notion fondamentale de calculabilité. p. 81, 83

Traitement automatique du langage naturel Domaine de l'intelligence artificielle qui se concentre sur la compréhension et la génération automatique de langage humain par les ordinateurs. p. 92

Traitement par lot (*batch processing*). Méthode de traitement informatique où un ensemble de tâches est exécuté en une seule fois, généralement sans intervention humaine directe. p. 207

Transformeur (*transformer*). Architecture de réseaux neuronaux conçue pour modéliser des relations complexes entre les éléments d'une séquence de données, tels que les mots dans un texte. Ils utilisent des mécanismes dits « d'attention » pour accorder une pondération variable aux différentes parties de la séquence lors de la prise de décision. p. 935, 936

Tri à bulles Algorithme de tri simple où les éléments sont comparés et échangés à répétition jusqu'à ce que toute la séquence soit triée, mais peu efficace pour les grandes quantités de données. p. 38

Turing-complet Propriété d'un langage de programmation ou d'un système capable de simuler une machine de Turing, ce qui signifie qu'il peut résoudre n'importe quel problème calculable. p. 149

Typage des données Processus de spécification du type de données associé à une variable ou une constante dans un langage de programmation, permettant la vérification *a priori* de la validité des expressions utilisant cette variable ou constante. p. 199

Type Catégorie ou classe de données définissant les valeurs possibles et les opérations applicables, utilisée pour garantir la cohérence et la sécurité dans les langages de programmation. p. 189

Type abstrait de données (*abstract data type*). Concept de programmation qui prédéfinit les opérations possibles sur les données, et par lesquelles seules ces données sont accessibles. Ce concept permet de dissocier la programmation de leur représentation par la machine lors de l'exécution du programme. p. 620

Variable globale Variable accessible et modifiable depuis n'importe quel endroit d'un programme, ce qui peut entraîner des problèmes de lisibilité et de maintenance du code. p. 130

Verbeux (*verbose*). Mode de communication entre un utilisateur et une application ou système, où ce dernier explicite au maximum ses opérations et son état ; un tel mode est parfois utile dans la résolution de problème de fonctionnement d'une application. p. 132

Vivacité (*liveliness*). Propriété d'un programme qu'une certaine situation (ou état) finira par se produire au cours de son éxecution. p. 844

Index des personnes

Bibliographie

Abelson, Harold, Gerald Jay Sussman et Julie Sussman (1985), *Structure and Interpretation of Computer Programs*, Cambridge, Mass : MIT Press.

Abrial, Jean-Raymond (2013), « From Z to B and Then Event-B : Assigning Proofs to Meaningful Programs », in : *Integrated Formal Methods*, sous la dir. d'Einar Broch Johnsen et Luigia Petre, Lecture Notes in Computer Science, Berlin : Springer, p. 1-15, doi : 10.1007/978-3-642-38613-8_1.

Académie Française [1694] (2001), *Dictionnaire de l'Académie Française (Première Édition)*, ARTFL, Chicago : University of Chicago, url : https://artfl.atilf.fr.

Acuña-Bravo, Wilber et al. (mai 2017), « Proportional Electro-Hydraulic Valves : An Embedded Model Control Solution », in : *Control Engineering Practice* 62, p. 22-35, doi : 10.1016/j.conengprac.2017.01.013.

Adriaans, Pieter (2013), « Information », in : *The Stanford Encyclopedia of Philosophy*, sous la dir. d'Edward N. Zalta, url : https://plato.stanford.edu/archives/fall2013/entries/information/ (visité le 24/01/2018).

Aho, Alfred V., Jeffrey D. Ullman et John E. Hopcroft [1974] (2006), *The Design and Analysis of Computer Algorithms*, Reading, Mass. : Pearson.

Alexandron, Giora et al. (oct. 2014), « Scenario-Based Programming, Usability-Oriented Perception », in : *ACM Transactions on Computing Education* 14.3, 21:1-21:23, doi : 10.1145/2648814.

Allen, James V. (2015), « Practical and Theoretical Knowledge in Aristotle », in : *Bridging the Gap between Aristotle's Science and Ethics*, sous la dir. d'Henry Devin et Karen Margrethe Nielsen, Cambridge : Cambridge University Press, p. 49-70.

Amershi, Saleema et al. (mai 2019), « Software Engineering for Machine Learning : A Case Study », in : *2019 IEEE/ACM 41st International Conference on*

Software Engineering : Software Engineering in Practice (ICSE-SEIP), p. 291-300, DOI : 10.1109/ICSE-SEIP.2019.00042.

ANAND, Pushkar (2021), *The Art of Writing Specifications in an Agile Ecosystem*, URL : https://modernanalyst.com/Resources/Articles/tabid/115/ID/5800/The-Art-of-Writing-Specifications-in-an-Agile-Ecosystem.aspx (visité le 17/10/2022).

ANDLER, Daniel (mai 2023), *Intelligence artificielle, intelligence humaine : la double énigme*, Paris : Gallimard.

ANDRICH, William et al. (fév. 2022), « Check and Validation of Building Information Models in Detailed Design Phase : A Check Flow to Pave the Way for BIM Based Renovation and Construction Processes », in : *Buildings* 12.2, p. 154, DOI : 10.3390/buildings12020154.

ANGIER, Tom P. S. (2010), *Technē in Aristotle's Ethics : Crafting the Moral Life*, Continuum Studies in Ancient Philosophy, London : Continuum.

ANGIUS, Nicola (2017), « From Simulative Programs as Theories to Theories of Simulative Programs », in : *Séminaire HEPIC*, Paris.

— (sept. 2019), « Qualitative Models in Computational Simulative Sciences : Representation, Confirmation, Experimentation. », in : *Minds and Machines* 29.3, p. 397-416.

ANSCOMBE, Gertrude Elizabeth Margaret (1981), *The Collected Philosophical Papers*, t. 2. Metaphysics and the Philosophy of Mind, Minneapolis : University of Minnesota Press.

— [1957] (sept. 2000), *Intention*, 2e éd., Cambridge, Mass : Harvard University Press.

ARENDT, Hannah [1958] (2012), « Condition de l'homme moderne », in : *L'humaine condition*, éd. établie par Philippe RAYNAUD, trad. par Georges FRADIER, Quarto, Paris : Gallimard, p. 53-323.

ARISTOTE (1985 –), *Aristoteteles et Corpus Aristotelicum*, éd. établie par Maria C. PANTELIA, Thesaurus Linguae Grecae Digital Library, Irvine, CA : University of California, URL : http://www.tlg.uci.edu (visité le 23/09/2023).

— (1984), *Complete Works of Aristotle : The Revised Oxford Translation*, éd. établie et trad. par Jonathan BARNES, 2 vol., Bollingen Series LXXI, Princeton : Princeton University Press.

— (2014a), *Œuvres : éthiques, politique, rhétorique, poétique, métaphysique*, éd. établie par Richard BODÉÜS, Bibliothèque de la Pléiade 601, Paris : Gallimard.

— (1950), *Organon*, trad. par Jules Tricot, 5 vol., Bibliothèque des textes philosophiques, Paris : Vrin.
— (2014b), *Premiers analytiques*, éd. établie et trad. par Michel Crubellier, GF, Paris : Flammarion.
— (2005), *Seconds Analytiques*, éd. établie et trad. par Pierre Pellegrin, GF, Paris : Flammarion.
— (2015), *Topiques ; Réfutations sophistiques*, éd. établie par Pierre Pellegrin, trad. par Jacques Brunschwig et Myriam Hecquet-Devienne, GF, Paris : Flammarion.
— (1960), *Posterior Analytics ; Topica*, trad. par E. S. Forster et Hugh Trennedick, Loeb Classical Library 391, Cambridge, Mass : Harvard University Press.
— [1973] (1994), *Posterior Analytics*, éd. établie et trad. par Jonathan Barnes, 2e éd., Clarendon Aristotle Series, Oxford : Clarendon Press.
— (1993), *Analytica Posteriora*, éd. établie par Ernst Grumach, trad. par Wolfgang Detel, 2 vol., Aristoteles Werke in deutscher Übersetzung, Berlin : Akademie-Verlag.
— (1987), *Ethique à Nicomaque*, trad. par Jules Tricot, 6e éd., 2 vol., Bibliothèque des textes philosophiques, Paris : Vrin.
— [1926] (1968), *Nicomachean Ethics*, trad. par Henry Rackham, Loeb Classical Library, Cambridge, Mass : Harvard University Press.
— (2014c), *Nicomachean Ethics*, trad. par Charles David Chanel Reeve, Indianapolis : Hackett Publishing Co.
— (1948), *Métaphysique*, trad. par Jules Tricot, 2e éd., 2 vol., Bibliothèque des textes philosophiques, Paris : Vrin.
— [1926] (1990), *Physique I-IV*, éd. établie et trad. par Henri Carteron, Collection des universités de France, Paris : Les Belles Lettres.
— (2013), *Le Mouvement des animaux - La Locomotion des animaux*, éd. établie par Pierre-Marie Morel, GF, Paris : Flammarion.
Aristote, pseudo- (juill. 2007), *The* Mechanical Problems *in the Corpus of Aristotle*, trad. par Thomas Winter, Lincoln : University of Nebraska, url : https://digitalcommons.unl.edu/classicsfacpub/68 (visité le 25/09/2023).
Arora, Daman, Himanshu Gaurav Singh et Mausam (mai 2023), *Have LLMs Advanced Enough ? A Challenging Problem Solving Benchmark For Large*

Language Models, DOI : 10.48550/arXiv.2305.15074, arXiv : 2305.15074 [cs].

ASHBY, William Ross [1956] (1957), *Introduction à la cybernétique*, trad. par Marcel PILLON, Paris : Dunod.

ASPER, Markus (2009), « The Two Cultures of Mathematics in Ancient Greece », in : *The Oxford Handbook of the History of Mathematics*, sous la dir. d'Eleanor ROBSON et Jacqueline STEDALL, Oxford : Oxford University Press, p. 107-132.

ASTARTE, Troy K. (juin 2019), « Formalising Meaning : A History of Programming Language Semantics », thèse de doctorat, Newcastle Upon Tyne.

AUBENQUE, Pierre [1962] (1972), *Le Problème de l'être chez Aristote : Essai sur la problématique aristotélicienne*, 3e éd., Bibliothèque de philosophie contemporaine, Paris : Presses universitaires de France.

AUSTIN, John Langshaw [1955] (1975), *How to Do Things with Words*, 2e éd., The William James Lectures, Oxford : Clarendon.

AYER, Alfred Jules [1936] (1952), *Language, Truth and Logic*, 2nd edition, reprint, New York : Dover.

BABBAGE, Charles [1822] (1989a), « The Science of Number Reduced to Mechanism », in : *The Works of Charles Babbage. 2 : The Difference Engine and Table Making*, The Pickering Masters, London : Pickering, p. 15-23.

— (1989b), *The Works of Charles Babbage. 3 : The Analytical Engine and Mechanical Notation*, sous la dir. de Martin CAMPBELL-KELLY, The Pickering Masters, London : Pickering.

— [1832] (2010), *On the Economy of Machinery and Manufactures*, Cambridge : Cambridge University Press, URL : https://www.cambridge.org/core/books/on-the-economy-of-machinery-and-manufactures/569622C494857D021A378AEE0D0E2E24 (visité le 02/12/2022).

BACHIMONT, Bruno (2004), « Critique de La Raison Computationnelle », URL : http://www.utc.fr/~bachimon/Publications_attachments/Bachimont.pdf.

— (oct. 2010), *Le Sens de la technique : le numérique et le calcul*, Paris : Encre Marine.

BACKUS, John (1978), « Can Programming Be Liberated from the von Neumann Style ? A Functional Style and Its Algebra of Programs », in : *Communications of the ACM* 21.8, p. 613-641.

Bacon, Francis [1620] (1986), *Novum organum*, trad. par Michel Malherbe et Jean-Marie Pousseur, Epiméthée, Paris : Presses universitaires de France.

Baier, Christel et Joost-Pieter Katoen (2008), *Principles of Model Checking*, Cambridge, Mass : MIT Press.

Bailer-Jones, Daniela M. (sept. 2002), « Scientists' Thoughts on Scientific Models », in : *Perspectives on Science* 10.3, p. 275-301, doi : 10.1162/106361402321899069.

Baltimore, David (2001), « How Biology Became an Information Science », in : *The Invisible Future : The Seamless Integration of Technology into Everyday Life*, p. 43-55.

Barendregt, Henk P. et Erik Barendsen [1984] (2000), *Introduction to Lambda Calculus*, url : https://www.cse.chalmers.se/research/group/logic/TypesSS05/Extra/geuvers.pdf (visité le 22/09/2023).

Barnes, Jonathan (1969), « Aristotle's Theory of Demonstration », in : *Phronesis* 14.2, p. 123-152.

— (1981), « Proof and Syllogism », in : *Aristotle on Science : The Posterior Analytics*, sous la dir. d'Enrico Berti, Università Di Padova, Padova : Editrice Antenore, p. 17-59.

— (2000), *Aristotle : A Very Short Introduction*, Oxford : Oxford University Press.

Beaujouan, Guy (1957), *L'interdépendance entre la science scolastique et les techniques utilitaires (XIIe, XIIIe et XIVe siècles)*, Les conférences du Palais de la Découverte, Paris : Palais de la Découverte.

Becerra, A. Gargantilla, M. Gutiérrez et R. Lahoz-Beltra (mars 2022), « Computing within Bacteria : Programming of Bacterial Behavior by Means of a Plasmid Encoding a Perceptron Neural Network », in : *Biosystems* 213, doi : 10.1016/j.biosystems.2022.104608.

Bechtel, William (2008), *Mental Mechanisms : Philosophical Perspectives on Cognitive Neuroscience*, New York : Routledge.

Bechtel, William et Adele Abrahamsen (sept. 2010), « Dynamic Mechanistic Explanation : Computational Modeling of Circadian Rhythms as an Exemplar for Cognitive Science », in : *Studies in History and Philosophy of Science Part A*, Computation and Cognitive Science 41.3, p. 321-333, doi : 10.1016/j.shpsa.2010.07.003.

Beck, Kent et al. (2001), *Manifesto for Agile Software Development*, url : https://agilemanifesto.org/iso/fr/manifesto.html (visité le 17/10/2022).

Beckmann, Johann [1806] (2017), *La technologie générale*, éd. établie par Guillaume Carnino, Liliane Hilaire-Pérez et Jochen Hoock, trad. par Joost Mertens, Histoire, Rennes : Presses universitaires de Rennes.

Beeson, Michael (2010), « Constructive Geometry », in : *Proceedings of the 10th Asian Logic Conference*, World Scientific, p. 19-84.

— (2015), « A Constructive Version of Tarski's Geometry », in : *Annals of Pure and Applied Logic* 166.11, p. 1199-1273.

Beisbart, Claus (mai 2018), « Are Computer Simulations Experiments ? And If Not, How Are They Related to Each Other ? », in : *European Journal for Philosophy of Science* 8.2, p. 171-204, doi : 10.1007/s13194-017-0181-5.

Bénis-Sinaceur, Hourya (mars 2000), « Address at the Princeton University Bicentennial Conference on Problems of Mathematics (December 17–19, 1946), By Alfred Tarski », in : *Bulletin of Symbolic Logic* 6.1, p. 1-44, doi : 10.2307/421074.

Bennett, Stuart (1993), *A History of Control Engineering, 1930-1955*, Control Engineering Series 47, Stevenage, UK : Peter Peregrinus / Institution of Electrical Engineers.

Berger, François et Jérémie Mattout (2017), *Interface cerveau-machine (ICM)*, url : https://www.inserm.fr/dossier/interface-cerveau-machine-icm/ (visité le 31/12/2022).

Bergstra, Jan Aldert (déc. 2021), « From Algorithms to Software Patents », in : *Colloque Definitions of Algorithms : From Logic to Law*, Université Paris 1 Panthéon-Sorbonne.

Bernard, Claude [1865] (2013), *Introduction à l'étude de la médecine expérimentale*, Champs Classiques, Paris : Flammarion.

Berry, Gérard (2008), *Pourquoi et comment le monde devient numérique*, Leçons inaugurales du Collège de France 197, Paris : Collège de France Fayard.

— (2017), *L'hyperpuissance de l'informatique : algorithmes, données, machines, réseaux*, Paris : Odile Jacob.

Berson, Ilene R. et Michael J. Berson, éd. (mai 2010), *High-Tech Tots : Childhood in a Digital World*, Information Age Publishing.

Bertalanffy, Ludwig von [1968] (juin 2015), *General System Theory : Foundations, Development, Applications*, New York : George Braziller.

Bjørner, Dines et Klaus Havelund (2014), « 40 Years of Formal Methods », in : *FM 2014 : Formal Methods*, sous la dir. de Cliff B. Jones, Pekka Pihlajasaari et Jun Sun, Lecture Notes in Computer Science, Cham : Springer, p. 42-61, doi : 10.1007/978-3-319-06410-9_4.

Blackwell, Alan F. (2002), « First Steps in Programming : A Rationale for Attention Investment Models », in : *Proceedings IEEE 2002 Symposia on Human Centric Computing Languages and Environments*, IEEE, p. 2-10.

Blair, Ann M. (nov. 2010), *Too Much to Know : Managing Scholarly Information before the Modern Age*, Yale University Press.

Blåsjö, Viktor (juin 2022), « Operationalism : An Interpretation of the Philosophy of Ancient Greek Geometry », in : *Foundations of Science* 27.2, p. 587-708, doi : 10.1007/s10699-021-09791-4.

Blass, Andreas, Nachum Dershowitz et Yuri Gurevich (2009), « When Are Two Algorithms the Same ? », in : *The Bulletin of Symbolic Logic* 15.2, p. 145-168.

Bocconi, Stefania et al. (juin 2016), « Developing Computational Thinking : Approaches and Orientations in K-12 Education », in : *EdMedia + Innovate Learning*, sous la dir. de George Veletsiano, Vancouver, BC, Canada : AACE.

Bongard, Joshua et Michael Levin (2021), « Living Things Are Not (20th Century) Machines : Updating Mechanism Metaphors in Light of the Modern Science of Machine Behavior », in : *Frontiers in Ecology and Evolution* 9.

Booch, Grady, Ivar Jacobson et James Rumbaugh (mai 2005), *The Unified Modeling Language User Guide*, 2e éd., Upper Saddle River, NJ : Addison Wesley.

Boolos, George S., John P. Burgess et Richard C. Jeffrey (2007), *Computability and Logic*, 5e éd., Cambridge : Cambridge University Press, doi : 10.1017/CBO9780511804076.

Boon, Mieke et Tarja Knuuttila (2009), « Models as Epistemic Tools in Engineering Sciences », in : *Philosophy of Technology and Engineering Sciences*, sous la dir. d'Anthonie W. M. Meijers, Paul Thagard et John Woods Jr., Amsterdam : Elsevier, p. 693-726, url : http://ebookcentral.proquest.com/lib/ensparis-ebooks/detail.action?docID=452811 (visité le 15/05/2018).

Bos, Henk J. M. (2001), *Redefining Geometrical Exactness : Descartes' Transformation of the Early Modern Concept of Construction*, New York : Springer.

Boucher, Claude (1972), *Leçons sur la théorie des automates mathématiques*, Berlin : Springer.

Bourdeau, Michel et Jean Mosconi, éd. (2022), *Anthologie de la calculabilité*, Paris : Cassini.

Bourdieu, Pierre (1980), *Le sens pratique*, Le Sens commun, Paris : Éditions de Minuit.

— [1972] (2000), *Esquisse d'une théorie de la pratique*, éd. revue par l'auteur, Points essais 405, Paris : Seuil.

— (nov. 2015), *Sociologie générale vol. 1 : Cours au Collège de France 1981-1983*, Paris : Seuil.

Bournez, Olivier et Amaury Pouly (2021), « A Survey on Analog Models of Computation », in : *Handbook of Computability and Complexity in Analysis*, sous la dir. de Vasco Brattka et Peter Hertling, Theory and Applications of Computability, Cham : Springer, p. 173-226, doi : 10.1007/978-3-030-59234-9_6.

Bowman, Judith S., Sandra L. Emerson et Marcy Darnovsky (1996), *The Practical SQL Handbook : Using Structured Query Language*, 3e éd., Reading, Mass : Addison-Wesley Developers Press.

Boyd, Iain D. et Thomas E. Schwartzentruber (mars 2017), *Nonequilibrium Gas Dynamics and Molecular Simulation*, Cambridge : Cambridge University Press.

Boyle, Robert (1674), *The Excellency of Theology Compar'd with Natural Philosophy*, London, url : http://name.umdl.umich.edu/A28966.0001.001.

Brading, Katherine (1999), « The Development of the Concept of Hypothesis from Copernicus to Boyle and Newton », in : *Revista de Filozofie KRISIS* 08-09, p. 5-16.

Bringsjord, Selmer (août 2015), « A Vindication of Program Verification », in : *History & Philosophy of Logic* 36.3, p. 262-277, doi : 10.1080/01445340.2015.1065461.

Bringsjord, Selmer et Michael John Zenzen (mars 2003), *Superminds : People Harness Hypercomputation, and More*, Berlin : Springer.

Brioist, Pascal (2008), « La réduction en art de l'escrime au XVIe siècle », in : *Réduire en art : la technologie de la Renaissance aux Lumières*, sous la dir. de Hélène Vérin et Pascal Dubourg Glatigny, Paris : Éditions de la Maison des sciences de l'homme, p. 238-257.

Brooks, Frederick P. (1982), *The Mythical Man-Month : Essays on Software Engineering*, Boston : Addison Wesley.

Brown, Frank M., éd. (1987), *The Frame Problem in Artificial Intelligence*, San Francisco : Morgan Kaufmann.

Brown, Tom B. et al. (juill. 2020), *Language Models Are Few-Shot Learners*, doi : 10.48550/arXiv.2005.14165.

Brunschwig, Jacques (1981), « L'objet et la structure des Seconds Analytiques d'après Aristote », in : *Aristotle on science : the Posterior analytics*, sous la dir. d'Enrico Berti, Università di Padova, Padova : Editrice Antenore, p. 61-96.

Bruyère, Nelly (1984), *Méthode et dialectique dans l'œuvre de La Ramée : Renaissance et âge classique*, De Pétrarque à Descartes 45, Paris : Vrin.

— (1986), « Le Statut de l'invention dans l'œuvre de La Ramée », in : *Revue des Sciences philosophiques et théologiques* 70.1, p. 15-24.

Buchanan, Richard (2009), « Thinking about Design : An Historical Perspective », in : *Philosophy of Technology and Engineering Sciences*, sous la dir. d'Anthonie W. M. Meijers, Paul Thagard et John Woods Jr., Amsterdam : Elsevier, p. 409-454, doi : 10.1016/B978-0-444-51667-1.50020-3.

Bunge, Mario (1966), « Technology as Applied Science », in : *Technology and Culture* 7.3, p. 329-347.

Burke, Tom [1994] (mai 1998), *Dewey's New Logic : A Reply to Russell*, New édition, Chicago : University of Chicago Press.

— (2002), « Mathematizing Dewey's Logical Theory », in : *Dewey's Logical Theory : New Studies and Interpretations*, sous la dir. de Tom Burke, D. Micah Hester et Robert B. Talisse, Vanderbilt University Press, p. 121-159.

Burke, Tom, D. Micah Hester et Robert B. Talisse, éd. (2002), *Dewey's Logical Theory : New Studies and Interpretations*, Vanderbilt University Press.

Butler, Michael et al. (2020), « The First Twenty-Five Years of Industrial Use of the B-method », in : *International Conference on Formal Methods for Industrial Critical Systems*, Berlin : Springer, p. 189-209.

Buzzard, Kevin (juill. 2020), « Formalising Undergraduate Mathematics », in : *CICM*, Imperial College, URL : https://www.ma.imperial.ac.uk/~buzzard/one_off_lectures/ug_maths.pdf (visité le 31/07/2023).

Calatayud, Agustina, John Mangan et M. Christopher (2019), « The Self-Thinking Supply Chain », in : *Supply Chain Management* 24.1, p. 22-38, DOI : 10.1108/SCM-03-2018-0136.

Caliste, Lisa et Guillaume Carnino (nov. 2022), « Qu'est-ce que l'industrie ? », in : *Artefact. Techniques, histoire et sciences humaines* 17, p. 219-242, DOI : 10.4000/artefact.13273.

Callaway, Ewen (2022), « What's next for the AI Protein-Folding Revolution », in : *Nature* 604, p. 234-238.

Camolezi, Marcos (fév. 2022), « Technique, Technics, Technik, substantifs : les mots à travers les dictionnaires du XIXe siècle », in : *Artefact. Techniques, histoire et sciences humaines* 15, p. 61-106, DOI : 10.4000/artefact.11313.

Campbell, Murray, A. Joseph Hoane et Feng-hsiung Hsu (jan. 2002), « Deep Blue », in : *Artificial Intelligence* 134.1, p. 57-83, DOI : 10.1016/S0004-3702(01)00129-1.

Campbell-Kelly, Martin et al. [1996] (2014), *Computer : A History of the Information Machine*, 3e éd., Boulder, Colorado : Westview Press, URL : http://archive.org/details/computerhistoryo0000camp_z9s0 (visité le 05/02/2023).

Capurro, Rafael (nov. 2009), « Past, Present, and Future of the Concept of Information », in : *tripleC : Communication, Capitalism & Critique. Open Access Journal for a Global Sustainable Information Society* 7.2, p. 125-141, DOI : 10.31269/triplec.v7i2.113.

Caracciolo, Alfonso (mars 1966), « Some Preliminary Remarks on Theoretical Pragmatics », in : *Communications of the ACM* 9.3, p. 226-227, DOI : 10.1145/365230.365273.

Carboncini, Sonia (1987), « L'encyclopédie et Christian Wolff : à propos de quelques articles anonymes », in : *Les études philosophiques* 4, p. 489-504.

Cardon, Dominique, Jean-Philippe Cointet et Antoine Mazières (2018), « La revanche des neurones : L'invention des machines inductives et la controverse de l'intelligence artificielle », in : *Réseaux* n° 211.5, p. 173, DOI : 10.3917/res.211.0173.

Carroll, Lewis (1895), « What the Tortoise Said to Achilles », in : *Mind* 4.14, p. 278-280.

Casilli, Antonio A. [2019] (sept. 2021), *En attendant les robots. Enquête sur le travail du clic*, Points, Paris : Seuil.

Castelvecchi, Davide (juin 2021), « Mathematicians Welcome Computer-Assisted Proof in 'Grand Unification' Theory », in : *Nature* 595.7865, p. 18-19, doi : 10.1038/d41586-021-01627-2.

Chalmers, David J. (sept. 2012), « The Varieties of Computation : A Reply », in : *Journal of Cognitive Science* 13.3, p. 211-248.

Chang, Hasok (2021), « Operationalism », in : *The Stanford Encyclopedia of Philosophy*, sous la dir. d'Edward N. Zalta, url : https://plato.stanford.edu/archives/fall2021/entries/operationalism/ (visité le 15/05/2023).

Charles, David (2015), « Aristotle on Practical and Theoretical Knowledge », in : *Bridging the Gap between Aristotle's Science and Ethics*, sous la dir. d'Henry Devin et Karen Margrethe Nielsen, Cambridge : Cambridge University Press, p. 71-93.

Châtillon, Jean (1965), « Le Didascalicon de Hugues de Saint-Victor », in : *Cahiers d'histoire mondiale* 9.1, p. 539-552.

Chun, Wendy Hui Kyong (2011), *Programmed Visions : Software and Memory*, Cambridge, Mass : MIT Press.

Clark, Andy (déc. 2010), *Supersizing the Mind. Embodiment, Action, and Cognitive Extension*, Oxford : Oxford University Press.

Clearsy (nov. 2021), *Formal Methods for Railways*, White Paper, Aix-en-Provence.

Cohen, I. Bernard (1969), « Hypotheses in Newton's Philosophy », in : *Proceedings of the Boston Colloquium for the Philosophy of Science 1966-1968*, sous la dir. de Robert S. Cohen et Marx W. Wartofsky, Boston Studies in the Philosophy of Science 5, Dordrecht : Springer, p. 304-326, doi : 10.1007/978-94-010-3381-7_8.

Colburn, Timothy (2000), *Philosophy and Computer Science*, Armonk, NY : M.E. Sharpe.

Condillac, Étienne Bonnot de [1749] (2010), *Traité Des Systèmes*, Chicoutimi : UQAC, url : http://dx.doi.org/doi:10.1522/030166753.

Constantinidis, Yves (2015), *Expression des besoins pour le SI : Guide d'élaboration du cahier des charges*, Paris : Eyrolles.

Courtine, Jean-Francois (jan. 1987), « Donner/Prendre : La Main », in : *Heidegger Studies* 3–4, p. 25-40.

Crandall, Richard L. (1988), *The Incorruptible Cashier*, Vestal, NY : Vestal Press, url : http://archive.org/details/incorruptiblecas0001cran (visité le 06/12/2022).

Craver, Carl F. et Lindley Darden (oct. 2013), *In Search of Mechanisms : Discoveries across the Life Sciences*, University of Chicago Press.

Crombie, Alistair Cameron (1994), *Styles of Scientific Thinking in the European Tradition : The History of Argument and Explanation Especially in the Mathematical and Biomedical Sciences and Arts*, 3 vol., London : Duckworth.

Cross, Nigel (2006), *Designerly Ways of Knowing*, London : Springer.

Crubellier, Michel (2014), « Premiers Analytiques : Introduction et appareil critique », in : *Premiers analytiques*, GF, Paris : Flammarion, p. 7-48, 235-397.

Crubellier, Michel et Pierre Pellegrin (2002), *Aristote : le philosophe et les savoirs*, Paris : Seuil.

Cunningham, Andrew (juill. 1982), « Two Legacies of the Later Alexandrian School : The Preliminary Questions in Commentaries ; the Theory/Practice Division of Medicine », in : *Nihon Ishigaku Zasshi. [Journal of Japanese History of Medicine]* 28.3, p. 400-424.

Date, Prasanna et al. (sept. 2022), « Neuromorphic Computing Is Turing-Complete », in : *Proceedings of the International Conference on Neuromorphic Systems 2022*, ICONS '22, New York : ACM Books, p. 1-10, doi : 10.1145/3546790.3546806.

Davis, Martin (2012), *The Universal Computer*, Turing Centenary Edition, Boca Raton : CRC Press.

De Millo, Richard A., Richard J. Lipton et Alan J. Perlis (1979), « Social Processes and Proofs of Theorems and Programs », in : *Communications of the ACM* 22.5, p. 271-280.

De Mol, Liesbeth (2012), « Looking for Busy Beavers. A Socio-Philosophical Study of a Computer-Assisted Proof », in : *Foundations of the Formal Sciences VII : Bringing Together Philosophy and Sociology of Science*, sous la dir. de Karen François et al., Studies in Logic 32, College Publications, p. 61-90.

— (2016), *Code Source sans Code : Le Cas de l'ENIAC*. Lille, URL : https://hal.science/cel-01345599/document.

De Mol, Liesbeth et Maarten Bullynck (avr. 2021), « Roots of 'program' Revisited », in : *Communications of the ACM* 64.4, p. 35-37, DOI : 10.1145/3419406.

— (2022), « What's in a Name ? Origins, Transpositions and Transformations of the Triptych Algorithm -Code -Program », in : *Abstractions and Embodiments : New Histories of Computing and Society*, sous la dir. de Janet Abbate et Stephanie Dick, Baltimore : John Hopkins Uuniversity Press, URL : https://hal.univ-lille.fr/hal-03081203.

De Pater, Wilhelmus Ant (1965), *Les Topiques d'Aristote et la dialectique platonicienne : La méthodologie de la définition*, Etudes thomistiques Vol. X, Fribourg, CH : Editions St. Paul.

De Risi, Vincenzo (août 2020), « Did Euclid Prove Elements I, 1 ? The Early Modern Debate on Intersections and Continuity », in : *Reading Mathematics in Early Modern Europe : Studies in the Production, Collection, and Use of Mathematical Books*, sous la dir. de Philip Beeley, Yelda Nasifoglu et Benjamin Wardhaugh, 1re éd., Routledge, p. 12-32, DOI : 10.4324/9781003102557, (visité le 21/07/2023).

Dean, Walter (2016), « Algorithms and the Mathematical Foundations of Computer Science », in : *Gödel's Disjunction : The Scope and Limits of Mathematical Knowledge*, p. 19-66.

Dear, Peter Robert (2006), *The Intelligibility of Nature : How Science Makes Sense of the World*, Science.Culture, Chicago : University of Chicago Press.

— (2019), *Revolutionizing the Sciences : European Knowledge in Transition, 1500-1700*, 3e éd., Oxford : Macmillan.

Dehaene, Stanislas [1995] (oct. 2010), *La Bosse des maths*, "Quinze ans après", Paris : Odile Jacob.

Dehaene, Stanislas, Yann Le Cun et Jacques Girardon [2018] (2020), *La plus belle histoire de l'intelligence : des origines aux neurones artificiels : vers une nouvelle étape de l'évolution*, Paris : Points Seuil.

Deldicque, Timothée (fév. 2022), « Canguilhem et « la technique » », in : *Artefact. Techniques, histoire et sciences humaines* 15, p. 201-238, DOI : 10.4000/artefact.11504.

Dennett, Daniel C. (oct. 2017), *Brainstorms : Philosophical Essays on Mind and Psychology*, Fortieth Anniversary Edition, Cambridge, Mass : MIT Press.

Denning, Peter J. (2004), « The Field of Programmers Myth », in : *Communications of the ACM* 47.7, p. 15-20.

Denning, Peter J. et Tim Bell (2012), « The Information Paradox », in : *American Scientist* 100.6, p. 470-477.

Denning, Peter J., Craig H. Martell et Vint Cerf (jan. 2015), *Great Principles of Computing*, Cambridge, Mass : MIT Press.

Depecker, Loïc (juin 2015), « Que diriez-vous d' « ordinateur » ? », in : *Bibnum. Textes fondateurs de la science*.

DeRemer, Frank et Hans Kron (avr. 1975), « Programming-in-the Large versus Programming-in-the-Small », in : *ACM SIGPLAN Notices* 10.6, p. 114-121, doi : 10.1145/390016.808431.

Descartes, René [1897-1911] (1996), *Œuvres*, éd. établie par Charles Adam et Paul Tannery, Paris : Vrin.

— (1994), *Œuvres Philosophiques*, éd. établie par Ferdinand Alquié, Classiques Garnier, Paris : Bordas.

— (1939), *Correspondance*, éd. établie par Charles Adam et Gérard Milhaud, 8 vol., Paris : Felix Alcan, puis Presses universitaires de France.

Desrosiers, Sophie (juin 2012), « Le textile structurel », in : *Techniques & Culture. Revue semestrielle d'anthropologie des techniques* 58, p. 82-103, doi : 10.4000/tc.6268.

Detlefsen, Michael (juin 2007), « Formalism », in : *The Oxford Handbook of Philosophy of Mathematics and Logic*, sous la dir. d'Edward Shapiro, Oxford : Oxford University Press, p. 51-74, doi : 10.1093/oxfordhb/9780195325928.001.0001.

Dewey, John (2008), *Collected Works*, éd. établie par Jo Ann Boydston, 38 vol., Carbondale : Southern Illinois university press.

— [1910] (1997), *How We Think*, Mineola, NY : Dover.

— [1925] (2012), *Expérience et nature*, trad. par Joëlle Zask, Bibliothèque de philosophie, Paris : Gallimard.

Dijksterhuis, Eduard Jan (1986), *The Mechanization of the World Picture : Phytagoras to Newton*, trad. par Carry Dikshoorn et Dirk Jan Struik, Princeton : Princeton University Press.

Dijksterhuis, Fokko Jan (2007), « Constructive Thinking : A Case for Dioptrics », in : *The Mindful Hand : Inquiry and Invention from the Late Renaissance to Early Industrialisation*, sous la dir. de Lissa Roberts, Simon Schaffer et Peter Robert Dear, History of Science and Scholarship in the Netherlands 9, Amsterdam : Koninklijke Nederlandse Akademie van Wetenschappen, p. 59-82.

Dijkstra, Edsger W. (1968), « Go to Statement Considered Harmful », in : *Communications of the ACM* 11.3, p. 147-148.

— (1974), « Programming as a Discipline of Mathematical Nature », in : *The American Mathematical Monthly* 81.6, p. 608-612.

— (mars 1976), *A Discipline of Programming*, Englewood Cliffs, N.J : Pearson.

— (1986), *Science Fiction and Science Reality in Computing (EWD 952)*, url : https://www.cs.utexas.edu/users/EWD/transcriptions/EWD09xx/EWD952.html (visité le 29/09/2022).

Dohmke, Thomas (fév. 2023), *GitHub Copilot for Business Is Now Available*, url : https://github.blog/2023-02-14-github-copilot-for-business-is-now-available/ (visité le 20/06/2023).

Dolza, Luisa et Hélène Vérin (2004), « Figurer la mécanique : l'énigme des théâtres de machines de la Renaissance », in : *Revue dhistoire moderne contemporaine* no51-2.2, p. 7-37.

Dominici, Gandolfo et Federica Palumbo (avr. 2013), « Decoding the Japanese Lean Production System According to a Viable Systems Perspective », in : *Systemic Practice and Action Research* 26.2, p. 153-171, doi : 10.1007/s11213-012-9242-z.

Domski, Mary (2009), « The Intelligibility of Motion and Construction : Descartes' Early Mathematics and Metaphysics, 1619–1637 », in : *Studies in History and Philosophy of Science Part A* 40.2, p. 119-130.

Dörner, Dietrich (1983), « Heuristics and Cognition in Complex Systems », in : *Methods of Heuristics*, sous la dir. de Rudolf Groner, Marina Groner et Walter F. Bischof, Hillsdale, NJ : Lawrence Erlbaum Associates, p. 89-107.

Dowek, Gilles (juin 2012), « The Physical Church Thesis as an Explanation of the Galileo Thesis », in : *Natural Computing* 11.2, p. 247-251, doi : 10.1007/s11047-011-9301-x.

Dreyfus, Hubert [1972] (jan. 1992), *Intelligence Artificielle. Mythes et limites*, trad. par Rose-Marie Vassallo-Villaneau et Daniel Andler, Paris : Flammarion.

Dubourg Glatigny, Pascal et Hélène Vérin (2008), « La réduction en art, un phénomène culturel », in : *Réduire en art : la technologie de la Renaissance aux Lumières*, sous la dir. de Hélène Vérin et Pascal Dubourg Glatigny, Paris : Éditions de la Maison des sciences de l'homme.

Dubucs, Jacques (2008), « Truth and Experience of Truth », in : *One Hundred Years of Intuitionism (1907-2007) : The Cerisy Conference*, sous la dir. de Mark van Atten et al., Basel : Springer, p. 50-59.

Dunne, Joseph [1993] (1997), *Back to the Rough Ground : Practical Judgment and the Lure of Technique*, sous la dir. d'Alasdair McIntyre, Paperback, Notre Dame, Ind. : University of Notre Dame Press.

Duns Scot, Jean (2017), *Questions sur la métaphysique*, éd. établie par Olivier Boulnois, trad. par Dan Arbib, Dominique Poirel et Olivier Boulnois, t. 1, Épiméthée, Paris : Presses universitaires de France.

Durand, Emmanuel (2012), « La Providence Du Salut Selon Thomas d'Aquin : Un Bénéfice Théologique de La Métaphysique Des Singuliers », in : *Revue des sciences philosophiques et théologiques* 96.3, p. 451-492.

Durand-Richard, Marie-José (1990), « Genèse de l'Algèbre Symbolique En Angleterre : Une Influence Possible de J. Locke », in : *Revue d'histoire des sciences* 43.2/3, p. 129-180.

— (déc. 2010), « Le regard français de Charles Babbage sur le « déclin de la science en Angleterre » », in : *Documents pour l'histoire des techniques. Nouvelle série* 19, p. 287-304.

Dutilh Novaes, Catarina (2011), « The Different Ways in Which Logic Is (Said to Be) Formal », in : *History and Philosophy of Logic* 32.4, p. 303-332.

Džeroski, Sašo et al. (fév. 1997), « Using Machine Learning Techniques in the Construction of Models. II. Data Analysis with Rule Induction », in : *Ecological Modelling* 95.1, p. 95-111, doi : 10.1016/S0304-3800(96)00029-4.

Ebbersmeyer, Sabrina (1995), « Spekulation », in : *Historisches Wörterbuch der Philosophie* 9, sous la dir. de Joachim Ritter et al., p. 1355-1371.

Edwards, Paul N. (oct. 1990), « The Army and the Microworld : Computers and the Politics of Gender Identity », in : *Signs : Journal of Women in Culture and Society* 16.1, p. 102-127, doi : 10.1086/494647.

Ensmenger, Nathan (juill. 2016), « The Multiple Meanings of a Flowchart », in : *Information & Culture* 51.3, p. 321-351, doi : 10.7560/IC51302.

Evans, Jonathan et Keith Frankish, éd. (jan. 2009), *In Two Minds : Dual Processes and Beyond*, Oxford : Oxford University Press.

Evans, Philip (juin 2000), « Strategy : The End to the End Game ? », in : *Journal of Business Strategy* 21.6, p. 12-16, doi : 10.1108/eb040124.

Fages, François et al. (2017), « Strong Turing Completeness of Continuous Chemical Reaction Networks and Compilation of Mixed Analog-Digital Programs », in : *Computational Methods in Systems Biology*, sous la dir. de Jérôme Feret et Heinz Koeppl, Lecture Notes in Computer Science, Cham : Springer, p. 108-127, doi : 10.1007/978-3-319-67471-1_7.

Falkenberg, Eckhard D. et al. (1998), *FRISCO : A framework of information system concepts : The FRISCO report (WEB edition)*, Laxenburg, Austria : International Federation for Information Processing (IFIP), url : https://research.utwente.nl/en/publications/frisco-a-framework-of-information-system-concepts-the-frisco-repo (visité le 12/02/2019).

Al-Farabi, Abu Nasr Muhammad (2015), *Le recensement des sciences*, éd. établie et trad. par Amor Cherni, Sagesses musulmanes, Beyrouth : Albouraq.

Fehlmann, Thomas et Eberhard Kranich (juill. 2020), « Intuitionism and Computer Science – Why Computer Scientists Do Not Like the Axiom of Choice », in : *Athens Journal of Sciences* 7, p. 143-158, doi : 10.30958/ajs.7-3-2.

Fetzer, James H. (1988), « Program Verification : The Very Idea », in : *Communications of the ACM* 31, p. 1048-1063, doi : 10.1145/48529.48530.

Ficin, Marsile [1482] (2001), *Platonic theology*, éd. établie par James Hankins et William Roy Bowen, trad. par Michael John Bridgman Allen et John Warden, The I Tatti Renaissance library 2, 4, 7, 13, 17, 23, Cambridge, Mass : Harvard University Press.

Finerty, J. Patrick (1979), « Cycles in Canadian Lynx », in : *The American Naturalist* 114.3, p. 453-455.

Fitzgerald, John et al. (2013), « Industrial Deployment of Formal Methods : Trends and Challenges », in : *Industrial Deployment of System Engineering Methods*, sous la dir. d'Alexander Romanovsky et Martyn Thomas, Heidelberg : Springer, p. 123-143.

Floridi, Luciano (2010), *Information : A Very Short Introduction*, Oxford : Oxford University Press.
— (jan. 2011), *The Philosophy of Information*, Oxford : Oxford University Press.
— (sept. 2017), « The Logic of Design as a Conceptual Logic of Information », in : *Minds and Machines* 27.3, p. 495-519, doi : 10.1007/s11023-017-9438-1.
— (sept. 2019a), « Marketing as Control of Human Interfaces and Its Political Exploitation », in : *Philosophy & Technology* 32.3, p. 379-388, doi : 10.1007/s13347-019-00374-7.
— (2019b), *The Logic of Information : A Theory of Philosophy as Conceptual Design*, First edition, Oxford : Oxford University Press.
Floyd, Christiane (1993), « Outline of a Paradigm Change in Software Engineering », in : *Program Verification : Fundamental Issues in Computer Science*, sous la dir. de Timothy R. Colburn, James H. Fetzer et Terry L. Rankin, Studies in Cognitive Systems 14, Dordrecht : Springer, p. 239-260.
Floyd, Juliet (2016), « Chains of Life : Turing, Lebensform, and the Emergence of Wittgenstein's Later Style », in : *Nordic Wittgenstein Review* 5.2, p. 7-89.
— (2018), « Lebensformen : Living Logic », in : *Language, Form(s) of Life, and Logic : Investigations After Wittgenstein*. Sous la dir. de Christian Martin, Berlin : Walter de Gruyter, p. 59-92.
Floyd, Robert W. (août 1979), « The Paradigms of Programming », in : *Communications of the ACM* 22.8, p. 455-460, doi : 10.1145/359138.359140.
Forman, Paul (mars 2007), « The Primacy of Science in Modernity, of Technology in Postmodernity, and of Ideology in the History of Technology », in : *History and Technology* 23.1-2, p. 1-152, doi : 10.1080/07341510601092191.
Forsythe, George E. (1959), « The Role of Numerical Analysis in an Undergraduate Program », in : *The American Mathematical Monthly* 66.8, p. 651-662, doi : 10.2307/2309339.
Fournier-Morel, Xavier et al. (2020), *SOA : microservices, API management : le guide de l'architecture d'un SI agile*, 5e éd., Malakoff : Dunod.
Frabetti, Federica (nov. 2014), *Software Theory : A Cultural and Philosophical Study*, London : Rowman & Littlefield Publishers.
Fraillon, Julian et al. (2020), *Preparing for Life in a Digital World : IEA International Computer and Information Literacy Study 2018*, Cham : Springer, doi : 10.1007/978-3-030-38781-5.

Friedman, Walter A. (1998), « John H. Patterson and the Sales Strategy of the National Cash Register Company, 1884 to 1922 », in : *Business History Review* 72.4, p. 552-584, doi : 10.2307/3116622.

Frigg, Roman et Julian Reiss (2009), « The Philosophy of Simulation : Hot New Issues or Same Old Stew ? », in : *Synthese* 169.3, p. 593-613.

Frost, Adam (jan. 2020), *4G Cellular V2I Communication with Traffic Lights Trial Kicks Off*, url : https://www.trafficTechnologyToday.com/news/connected-vehicles-infrastructure/4g-cellular-v2i-communication-with-traffic-lights-trial-kicks-off.html (visité le 07/02/2022).

Furetière, Antoine (1701), *Dictionnaire universel*, 2e éd., 3 vol., La Haye, url : https://gallica.bnf.fr/ark:/12148/bpt6k56749155 (visité le 27/08/2021).

Gabbay, John et Andrée Le May (2010), *Practice-Based Evidence for Healthcare : Clinical Mindlines*, Routledge.

— (2016), *Mindlines : Making Sense of Evidence in Practice*.

Gabbey, Alan (1984), « The Mechanical Philosophy and Its Problems : Mechanical Explanations, Impenetrability, and Perpetual Moti », in : *Change and Progress in Modern Science*, sous la dir. de Joseph C. Pitt, Dordrecht : Springer, p. 9-84, doi : 10.1007/978-94-009-6525-6.

— (1992), « Newton's Mathematical Principles of Natural Philosophy : A Treatise on 'Mechanics' ? », in : *The Investigation of Difficult Things : Essays on Newton and the History of the Exact Sciences in Honour of D.T. Whiteside*, sous la dir. de Peter Michael Harman et Alan E. Shapiro, Cambridge : Cambridge University Press, p. 305-322.

— (1996), « Newton and Natural Philosophy », in : *Companion to the History of Modern Science*, Routledge.

— (2001), « Mechanical Philosophies and Their Explanations », in : *Late Medieval and Early Modern Corpuscular Matter Theories*, sous la dir. de Christoph Lüthy, John Emery Murdoch et William Royall Newman, Medieval and Early Modern Science 1, Leiden : Brill, p. 441-466.

— (2004), « What Was "Mechanical" about "The Mechanical Philosophy" ? », in : *The Reception of the Galilean Science of Motion in Seventeenth-Century Europe*, sous la dir. de Carla Rita Palmerino et al., Boston Studies in the

Philosophy of Science 239, Dordrecht : Springer, p. 11-23, DOI : 10.1007/978-1-4020-2455-9, (visité le 06/03/2023).

GALILEI, Galileo [1634] (1966), *Les méchaniques de Galilée, mathématicien et ingénieur du duc de Florence*, éd. établie par Bernard ROCHOT, trad. par Marin MERSENNE, Le Mouvement des idées au XVIIe siècle 4, Paris : Presses universitaires de France.

GAMMA, Erich et al. (oct. 1994), *Design Patterns : Elements of Reusable Object-Oriented Software*, Addison-Wesley.

GANDY, Robin (1980), « Church's Thesis and Principles for Mechanisms », in : *The Kleene Symposium*, sous la dir. de Jon BARWISE et H. Jerome KEISLER, North-Holland, p. 123-148.

GARAPON, Antoine et Jean LASSÈGUE (2018), *Justice digitale : révolution graphique et rupture anthropologique*, Paris : Presses universitaires de France.

GARBER, Daniel (2013), « Remarks on the Pre-history of the Mechanical Philosophy », in : *The Mechanization of Natural Philosophy*, sous la dir. de Daniel GARBER et Sophie ROUX, Boston Studies in the Philosophy of Science 300, Dordrecht : Springer, p. 3-26.

GARBER, Daniel et Sophie ROUX, éd. (2013), *The Mechanization of Natural Philosophy*, Boston Studies in the Philosophy of Science 300, Dordrecht : Springer.

GARÇON, Anne-Françoise (2008), « Réduire la mine en science ?... Anatomie des De re metallica d'Agricola », in : *Réduire en art : la technologie de la Renaissance aux Lumières*, sous la dir. de Hélène VÉRIN et Pascal DUBOURG GLATIGNY, Paris : Éditions de la Maison des sciences de l'homme.

GATCOMB, P. (nov. 2021), *C : MO - Using LUA to Simulate IADS Disruption*, URL : https://www.youtube.com/watch?v=vuBIUiP0eLU (visité le 22/01/2023).

GAUDILLIÈRE-JAMI, Nadja (2022), « Automatiser l'Architecture », thèse de doctorat, Paris : Université Paris-Est.

GAVARD, Emmanuel (juill. 2023), « La déconnexion, un geste politique », in : *Stratégies*, p. 20-21.

GERBERT, Philipp et al. (2017), *Powering the Service Economy with RPA and AI*, rapp. tech., Boston, URL : https://www.bcg.com/publications/2017/technology-digital-operations-powering-the-service-economy-with-rpa-ai (visité le 10/10/2018).

Giacomo, Luchetta et al. (2022), *VAT in the Digital Age : Executive Summary*. Rapp. tech. TAXUD/2019/CC/150, Luxembourg, URL : https://data.europa.eu/doi/10.2778/141964 (visité le 06/08/2023).

Giglioni, Guido (2022), « The Age of the New Organons : Ramus, Bacon and Descartes, and the Making of Thinking Tools in the Early Modern Period (1550-1650) », in : *Conférence : Anciens et Nouveaux Organons Techniques de Savoir Entre XVIe et XVIIe Siècles*, Université Paris 1 Panthéon-Sorbonne.

Gilbert, Neal Ward (1960), *Renaissance Concepts of Method*, New York : Columbia University press.

Goldin, Dina (fév. 2000), « Persistent Turing Machines as a Model of Interactive Computation », in : p. 116-135, DOI : 10.1007/3-540-46564-2_8.

Goldin, Dina et al. (nov. 2004), « Turing Machines, Transition Systems, and Interaction », in : *Information and Computation* 194.2, p. 101-128, DOI : 10.1016/j.ic.2004.07.002.

Goldschmidt, Victor [1953] (1989), *Le systeme stoïcien et l'idée de temps*, 4e éd., Bibliothèque d'histoire de la philosophie, Paris : Vrin.

Goodfellow, Ian, Yoshua Bengio et Aaron Courville (2016), *Deep Learning*, Adaptive Computation and Machine Learning, Cambridge, Mass : MIT Press.

Gorn, Saul (1963), « The Computer and Information Sciences : A New Basic Discipline », in : *SIAM Review* 5.2, p. 150-155.

— (1968), « The Identification of the Computer and Information Sciences : Their Fundamental Semiotic Concepts and Relationships », in : *Foundations of language*, p. 339-372.

Grabowska, Sandra (juin 2020), « Smart Factories in the Age of Industry 4.0 », in : *Management Systems in Production Engineering* 28.2, p. 90-96, DOI : 10.2478/mspe-2020-0014.

Graça, Daniel Silva et José Félix Costa (2003), « Analog Computers and Recursive Functions over the Reals », in : *Journal of Complexity* 19.5, p. 644-664.

Granström, Johan Georg (juin 2011), *Treatise on Intuitionistic Type Theory*, Berlin : Springer.

Grattan-Guinness, Ian (juill. 1990), « Work for the Hairdressers : The Production of de Prony's Logarithmic and Trigonometric Tables », in : *Annals of the History of Computing* 12.3, p. 177-185, DOI : 10.1109/MAHC.1990.10029.

Green, Thomas R. G. (1990a), « Programming Languages as Information Structures », in : *Psychology of Programming*, sous la dir. de J.-M. Hoc et al., Academic Press, p. 117-137.

— (1990b), « The Nature of Programming », in : *Psychology of Programming*, sous la dir. de J.-M. Hoc et al., Academic Press, p. 21-44.

Greiler, Michaela (déc. 2020), *"Code Reviews – From Bottleneck to Superpower" with Michaela Greiler*, url : https://www.youtube.com/watch?v=gRR-UhusQe8 (visité le 05/01/2023).

Grene, Marjorie (1995), « Animal Mechanism and the Cartesian Vision of Nature », in : *Physics, Philosophy, and the Scientific Community : Essays in Honor of Robert S. Cohen*, sous la dir. de Kostas Gavroglu, John Stachel et Marx W. Wartofsky, Boston Studies in the Philosophy of Science, Dordrecht : Springer, p. 189-204, doi : 10.1007/978-94-017-2658-0_10.

Grier, David Alan (2005), *When Computers Were Human*, Princeton : Princeton University Press.

Gryz, Jarek (2013), « The Frame Problem in Artificial Intelligence and Philosophy », in : *Filozofia Nauki* 21.2 (82), p. 15-30.

Guillerme, Jacques et Jan Sebestik [1968] (déc. 2007), « Les commencements de la technologie », in : *Documents pour l'histoire des techniques. Nouvelle série* 14, p. 49-122.

Gunter, Carl A. et al. (2000), « A Reference Model for Requirements and Specifications », in : *IEEE Software* 17.3, p. 37-43.

Gupta, Anil (2015), « Definitions », in : *The Stanford Encyclopedia of Philosophy*, sous la dir. d'Edward N. Zalta, url : https://plato.stanford.edu/archives/sum2015/entries/definitions/ (visité le 05/01/2019).

Gurevich, Yuri et Marc Spielmann (1997), « Recursive Abstract State Machines », in : *Journal of Universal Computer Science* 3.4, p. 233-246.

Hacking, Ian [1975] (2006), *The Emergence of Probability : A Philosophical Study Ofearly Ideas about Probability, Induction and Statistical Inference*, 2e éd., Cambridge : Cambridge University Press.

— (2014), *Why Is There Philosophy of Mathematics at All ?*, Cambridge : Cambridge University Press.

Hager, F.-P. et Ch. Strub (1995), « System - Systematik - Systematisch », in : *Historisches Wörterbuch der Philosophie* 10, sous la dir. de Joachim Ritter et al., p. 824-856.

Haigh, Thomas et Paul E. Ceruzzi (sept. 2021), *A New History of Modern Computing*, Cambridge, Mass : MIT Press.

Haigh, Thomas, Mark Priestley et Crispin Rope (juill. 2014), « Los Alamos Bets on ENIAC : Nuclear Monte Carlo Simulations, 1947-1948 », in : *IEEE Annals of the History of Computing* 36.3, p. 42-63, doi : 10.1109/MAHC.2014.40.

Hall, Anthony (2010), « E = Mc^2 Explained », in : *Software Requirements and Design : The Work of Michael Jackson*, sous la dir. de Bashar Nuseibeh et Pamela Zave, Chatham, NJ : Good Friends Publishing.

Hall, Mark [2008] (2017), « Amazon.Com », in : *Encyclopedia Britannica*, url : https://www.britannica.com/topic/Amazoncom (visité le 16/01/2023).

Hamblin, Charles L. (1976), « An Improved Pons Asinorum ? », in : *Journal of the History of Philosophy* 14.2, p. 131-136, doi : 10.1353/hph.2008.0365.

Hamou, Philippe (sept. 2014), « Sur les origines du concept de méthode à l'âge classique : La Ramée, Bacon et Descartes », in : *LISA* vol. XII-n°5, doi : 10.4000/lisa.6249.

Hansen, Jörg (2008), « Is There a Logic of Imperatives ? », in : *Deontic Logic in Computer Science*, Citeseer.

Harel, David, Dexter Kozen et Jerzy Tiuryn (sept. 2000), *Dynamic Logic*, Cambridge, Mass : MIT Press.

Harnad, Stevan (1990), « The Symbol Grounding Problem », in : *Physica D : Nonlinear Phenomena* 42.1-3, p. 335-346.

— (1993), « Problems, Problems : The Frame Problem as a Symptom of the Symbol Grounding Problem », in : *Psycoloquy* 4.34.

Hartmanis, Juris (jan. 1981), « Observations About the Development of Theoretical Computer Science », in : *Annals of the History of Computing* 3.1, p. 42-51, doi : 10.1109/MAHC.1981.10005.

— (1995), « On Computational Complexity and the Nature of Computer Science », in : *ACM Computing Surveys* 27.1, p. 7-16.

Hayes, Ian James et Cliff B. Jones (1989), « Specifications Are Not (Necessarily) Executable », in : *Software Engineering Journal* 4.6, p. 330-339.

Heidegger, Martin (nov. 1990), *Questions III et IV*, Tel 172, Paris : Gallimard.

Heidegger, Martin [1927] (1993), *Sein und Zeit*, Tübingen : Max Niemeyer.

— [1957] (2003), *Le principe de raison*, trad. par André Préau, Tel 79, Paris : Gallimard.

Hempel, Carl Gustav (1965), *Aspects of Scientific Explanation : And Other Essays in the Philosophy of Science*, New York : The Free Press Collier-Macmillan.

Hendricks, Vincent Fella, Arne Jakobsen et Stig Andur Pedersen (2000), « Identification of Matrices in Science and Engineering », in : *Journal for General Philosophy of Science* 31.2, p. 277-305.

Hersh, Reuben (1997), *What Is Mathematics, Really ?*, New York : Oxford University Press.

Herzig, Andreas (2015), « Logics of Knowledge and Action : Critical Analysis and Challenges », in : *Autonomous Agents and Multi-Agent Systems* 29, p. 719-753.

Heule, Marijn J. H. et Oliver Kullmann (juill. 2017), « The Science of Brute Force », in : *Communications of the ACM* 60.8, p. 70-79, doi : 10.1145/3107239.

Hewitt, Carl, Peter Boehler Bishop et Richard Steiger (août 1973), « A Universal Modular ACTOR Formalism for Artificial Intelligence », in : *Proceedings of the 3rd International Joint Conference on Artificial Intelligence*, Stanford : IJCAI, p. 235-245, url : http://ijcai.org/Proceedings/73/Papers/027B.pdf.

Heyting, Arend (déc. 1931), « Die intuitionistische Grundlegung der Mathematik », in : *Erkenntnis* 2.1, p. 106-115, doi : 10.1007/BF02028143.

Hickman, Larry A. (1992), *John Dewey's Pragmatic Technology*, 1st Midland book ed, The Indiana Series in the Philosophy of Technology 763, Bloomington : Indiana University Press.

Hilbert, David [1922] (1996), « The New Grounding of Mathematics. First Report », in : *From Kant to Hilbert : A Source Book in the Foundations of Mathematics*, sous la dir. de William Bragg Ewald, t. 2, Oxford : Clarendon Press, p. 1115-34.

Hildebrand, David L. (2008), *Dewey : A Beginner's Guide*, Oneworld Beginner's Guides, Oxford : Oneworld.

Hoare, Tony (1969), « An Axiomatic Basis for Computer Programming », in : *Communications of the ACM* 12.10, p. 576-583.

— (août 1986), « Mathematics of Programming », in : *BYTE* 11.8, p. 115-149.
— (1996), « How Did Software Get so Reliable without Proof ? », in : *FME'96 : Industrial Benefit and Advances in Formal Methods*, sous la dir. de Marie-Claude Gaudel et James Woodcock, Lecture Notes in Computer Science, Berlin : Springer, p. 1-17.
Hoffrage, Ulrich et al. (juill. 2002), « Representation Facilitates Reasoning : What Natural Frequencies Are and What They Are Not », in : *Cognition* 84.3, p. 343-352, doi : 10.1016/S0010-0277(02)00050-1.
Hofmann, Peter, Caroline Samp et Nils Urbach (mars 2020), « Robotic Process Automation », in : *Electronic Markets* 30.1, p. 99-106, doi : 10.1007/s12525-019-00365-8.
Hofstadter, Douglas R. [1979] (1999), *Gödel, Escher, Bach : An Eternal Golden Braid*, 20th anniversary, New York : Basic Books.
Hollan, James, Edwin Hutchins et David Kirsh (juin 2000), « Distributed Cognition : Toward a New Foundation for Human-Computer Interaction Research », in : *ACM Transactions on Computer-Human Interaction* 7.2, p. 174-196, doi : 10.1145/353485.353487.
Holt, Anatol W. (1997), *Organized Activity and Its Support by Computer*, Berlin : Springer.
Holt, Anatol W. et Felice Cardone (sept. 1999), « An Organizational Theory of Information », in : *Information System Concepts : An Integrated Discipline Emerging*, sous la dir. d'Eckhard D. Falkenberg, Kalle Lyytinen et Alexander A. Verrijn-Stuart, Leiden : Springer, p. 77-91, doi : 10.1007/978-0-387-35500-9_6.
Hood, Colin et al., éd. (2008), *Requirements Management : The Interface between Requirements Development and All Other Systems Engineering Processes*, Berlin : Springer.
Hooker, Cliff A. et al., éd. (2011), *Philosophy of Complex Systems*, Handbook of the Philosophy of Science 10, Amsterdam : Elsevier.
Houkes, Wybo (2009), « The Nature of Technological Knowledge », in : *Philosophy of Technology and Engineering Sciences*, sous la dir. d'Anthonie W. M. Meijers, Paul Thagard et John Woods Jr., Amsterdam : Elsevier.
Houkes, Wybo et Pieter E. Vermaas (2004), « Actions Versus Functions : A Plea for an Alternative Metaphysics of Artifacts », in : *The Monist* 87.1, p. 52-71.

Houkes, Wybo, Pieter E. Vermaas et al. (mai 2002), « Design and Use as Plans : An Action-Theoretical Account », in : *Design Studies*, Philosophy of Design 23.3, p. 303-320, doi : 10.1016/S0142-694X(01)00040-0.

Hubin, Jean-Benoît (2018), « La responsabilité du fait des robots : le droit de la responsabilité à l'ère de la révolution numérique », in : *Responsabilités et numérique*, Jeune barreau de Namur, Limal : Anthemis, p. 257-280.

Hügli, Anton et U. Theissmann (1971), « Invention, Erfindung, Entdeckung », in : *Historisches Wörterbuch der Philosophie* 4, sous la dir. de Joachim Ritter et al., p. 544-573.

Hume, David [1740] (1990), *A Treatise of Human Nature*, éd. établie par Lewis A. Selby-Bigge et al., 2e éd., Oxford : Clarendon Press.

Humphreys, Paul [2004] (avr. 2007), *Extending Ourselves : Computational Science, Empiricism, and Scientific Method*, new, Oxford : Oxford University Press.

— (2020), « Why Automated Science Should Be Cautiously Welcomed », in : *A Critical Reflection on Automated Science : Will Science Remain Human ?*, sous la dir. de Marta Bertolaso et Fabio Sterpetti, Human Perspectives in Health Sciences and Technology, Cham : Springer, p. 11-26, doi : 10.1007/978-3-030-25001-0_2.

Hutchins, Edwin (1995), *Cognition in the Wild*, Cambridge, Mass : MIT Press.

— (2010), « Cognitive Ecology », in : *Topics in Cognitive Science* 2.4, p. 705-715, doi : 10.1111/j.1756-8765.2010.01089.x.

Jackson, Daniel (2006), *Software Abstractions : Logic, Language and Analysis*, Cambridge, Mass : MIT Press.

— (2021), *The Essence of Software : Why Concepts Matter for Great Design*, Princeton : Princeton University Press.

— (juill. 2023), *LLM Coders : The End of Agile ?*, The Essence of Software, url : https://essenceofsoftware.com/posts/llm-coding/ (visité le 02/08/2023).

Jackson, Michael (2001a), « Problem Analysis and Structure », in : *Nato Science Series Sub Series III Computer and Systems Sciences* 180, p. 3-20.

— (2001b), *Problem Frames : Analysing and Structuring Software Development Problems*, Edinburgh : Addison-Wesley.

— (sept. 2002), « Some Basic Tenets of Description », in : *Software and Systems Modeling* 1.1, p. 5-9, doi : 10.1007/s10270-002-0005-7.

— (2005), « Problem Frames and Software Engineering », in : *Information and Software Technology* 47.14, p. 903-912.
— (2010), « Program Verification and System Dependability », in : *Formal Methods : State of the Art and New Directions*, sous la dir. de Paul Boca, Jonathan P. Bowen et Jawed Siddiqi, London : Springer, doi : 10.1007/978-1-84882-736-3.
Jacobs, F. Robert et F. C. 'Ted' Weston (mars 2007), « Enterprise Resource Planning (ERP)—A Brief History », in : *Journal of Operations Management* 25.2, p. 357-363, doi : 10.1016/j.jom.2006.11.005.
Jacobson, Ivar et al. (2019), *The Essentials of Modern Software Engineering : Free the Practices from the Method Prisons !*, ACM Books.
Jardine, Nicholas (1976), « Galileo's Road to Truth and the Demonstrative Regress », in : *Studies in History and Philosophy of Science Part A* 7.4, p. 277-318.
Jauréguiberry, Francis (2014), « La déconnexion aux technologies de communication », in : *Réseaux* 186.4, p. 15-49, doi : 10.3917/res.186.0015.
Johansen, Thomas Kjeller (2017), « Aristotle on the Logos of the Craftsman », in : *Phronesis* 62.2, p. 97-135.
Johnston, James Scott (nov. 2014), *John Dewey's Earlier Logical Theory*, New York : SUNY Press.
— (2020), *John Dewey's Later Logical Theory*, American Philosophy and Cultural Though, New York : SUNY Press.
Jones, Cliff B. et Troy K. Astarte (2017), « Challenges for Semantic Description : Comparing Responses from the Main Approaches », in : *School of Computing Science Technical Report Series*.
— (2018), « Challenges for Formal Semantic Description : Responses from the Main Approaches », in : *Engineering Trustworthy Software Systems*, sous la dir. de Jonathan P. Bowen, Zhiming Liu et Zili Zhang, Cham : Springer, p. 176-217, doi : 10.1007/978-3-030-02928-9_6.
Jones, David et al. (mai 2020), « Characterising the Digital Twin : A Systematic Literature Review », in : *CIRP Journal of Manufacturing Science and Technology* 29, p. 36-52, doi : 10.1016/j.cirpj.2020.02.002.
Joyce, David (1998), *Euclid's Elements*, url : http://aleph0.clarku.edu/~djoyce/java/elements/elements.html (visité le 06/10/2021).

Jürgensen, Helmut (fév. 2012), « Invariance and Universality of Complexity », in : *Computation, Physics and Beyond*, sous la dir. de Michael J. Dinneen, Bakhadyr Khoussainov et André Nies, Lecture Notes in Computer Science, Auckland, New Zealand : Springer, p. 140-158, doi : 10.1007/978-3-642-27654-5_11.

Kahneman, Daniel (mai 2012), *Thinking, Fast and Slow*, London : Penguin.

Kant, Emmanuel (1902), *Gesammelte Schriften*, Akademie Ausgabe, Bonn.

— [1790] (1986), *Œuvres philosophiques*, éd. établie par Ferdinand Alquié, Bibliothèque de la Pléiade, 3 vol., Paris : Gallimard.

— [1785] (1987), *Fondements de la metaphysique des moeurs*, trad. par Victor Delbos et Alexis Philonenko, Paris : Vrin.

— [1789] (1989), *Critique de la faculté de Juger*, trad. par Alexis Philonenko, Paris : Vrin.

Kaplan, Craig A. et Herbert A. Simon (juill. 1990), « In Search of Insight », in : *Cognitive Psychology* 22.3, p. 374-419, doi : 10.1016/0010-0285(90)90008-R.

Kaspar, Corinna et al. (juin 2021), « The Rise of Intelligent Matter », in : *Nature* 594.7863, p. 345-355, doi : 10.1038/s41586-021-03453-y.

Kay, Alan C. (1996), « The Early History of Smalltalk », in : *History of Programming Languages—II*, New York : ACM Books, p. 511-598, doi : 10.1145/234286.1057828.

Kay, Alan C. et Stefan L. Ram (juill. 2003), *On the Meaning of "Object-Oriented Programming"*, url : http://www.purl.org/stefan_ram/pub/doc_kay_oop_en (visité le 06/07/2020).

Kelleher, Caitlin et Randy Pausch (juin 2005), « Lowering the Barriers to Programming : A Taxonomy of Programming Environments and Languages for Novice Programmers », in : *ACM Computing Surveys* 37.2, p. 83-137, doi : 10.1145/1089733.1089734.

Kerstiens, Ludwig (1958), « Die Lehre von Der Theoretischen Erkenntnis in Der Lateinischen Tradition », in : *Philosophisches Jahrbuch* 66, p. 375-424.

Kiran, D. R. (jan. 2017), « Seven Traditional Tools of TQM », in : *Total Quality Management*, sous la dir. de D. R. Kiran, Butterworth-Heinemann, p. 271-290, doi : 10.1016/B978-0-12-811035-5.00020-9.

Klein, John, Harry Levinson et Jay Marchetti (2015), *Model-Driven Engineering : Automatic Code Generation and Beyond*, Technical Report CMU/SEI-

2015-TN-005, Pittsburgh, chap. Technical Reports, URL : https://apps.dtic.mil/sti/citations/AD1046652 (visité le 17/10/2022).

KNECHT, Herbert H. (1981), *La logique chez Leibniz : essai sur le rationalisme baroque*, Lausanne : L'âge d'homme.

KNORR, Wilbur R. (1983), « Construction as Existence Proof in Ancient Geometry », in : *Ancient Philosophy* 3, p. 125-148.

— (1986), *The Ancient Tradition of Geometric Problems*, Boston : Birkhäuser.

— (déc. 1989), « The Practical Element in Ancient Exact Sciences », in : *Synthese* 81.3, p. 313-328, DOI : 10.1007/BF00869319.

KNUTH, Donald E. (1974), « Computer Science and Its Relation to Mathematics », in : *The American Mathematical Monthly* 81.4, p. 323-343.

— (1977), « Algorithms », in : *Scientific American* 236.4, p. 63-81.

— (mars 1985), « Algorithmic Thinking and Mathematical Thinking », in : *The American Mathematical Monthly* 92.3, p. 170-181, DOI : 10.1080/00029890.1985.11971572.

— (juill. 1997), *The Art of Computer Programming*, 3e éd., t. Vol. 1 : Fundamental Algorithms, Reading, Mass : Addison-Wesley.

KOLMOGOROV, Andreï (déc. 1932), « Zur Deutung der intuitionistischen Logik », in : *Mathematische Zeitschrift* 35.1, p. 58-65, DOI : 10.1007/BF01186549.

KÖNIG, Gert et Helmut PULTE (1998), « Theorie », in : *Historisches Wörterbuch der Philosophie* 10, sous la dir. de Joachim RITTER et al., p. 1128-1154.

KOSTIC, Neven (2009), « Topologie Des Champs de Contraintes Pour Le Dimensionnement Des Structures En Beton Arme », thèse de doctorat, Lausanne : EPFL.

KOYDAN, Kateryna (avr. 2020), *Is SQL a Programming Language ?*, URL : https://learnsql.com/blog/sql-programming-language/ (visité le 21/01/2023).

KOYRÉ, Alexandre (1961), *Études d'histoire de la pensée philosophique*, Paris : Armand Colin.

KROES, Peter A. (2012), *Technical Artefacts : Creations of Mind and Matter : A Philosophy of Engineering Design*, Philosophy of Engineering and Technology 6, Dordrecht : Springer.

KULKARNI, Deepak et Herbert A. SIMON (avr. 1988), « The Processes of Scientific Discovery : The Strategy of Experimentation », in : *Cognitive Science* 12.2, p. 139-175, DOI : 10.1207/s15516709cog1202_1.

La Ramée, Pierre de [1555] (1996), *Dialectique 1555 : un manifeste de la Pléiade*, éd. établie par Nelly Bruyère, De Pétrarque à Descartes 61, Paris : Vrin.

Laird, Walter Roy et Sophie Roux, éd. (2008), *Mechanics and Natural Philosophy before the Scientific Revolution*, Boston Studies in the Philosophy of Science volume 254, Dordrecht : Springer.

Lakatos, Imre [1986] (1998), « A Renaissance of Empiricism », in : *New Directions in the Philosophy of Mathematics : An Anthology*, sous la dir. de Thomas Tymoczko, revised and expanded paperback ed., Princeton : Princeton University Press, p. 29-48.

Lalande, André [1926] (1992), *Vocabulaire technique et critique de la philosophie*, Quadrige, Paris : Presses universitaires de France.

Lambert, Johann Heinrich (1764), *Neues Organon*, Leipzig : Wendler.

— (1988), *Texte zur Systematologie und zur Theorie der wissenschftlichen Erkenntnis*, éd. établie par Geo Siegwart, Philosophische Bibliothek 406, Hamburg : Felix Meiner Verlag.

Lando, Pascal et al. (2007), « Premiers Pas Vers Une Ontologie Générale Des Programmes Informatiques », in : *18es Journées Francophones d'Ingénierie Des Connaissances*, Grenoble, URL : https://hal.archives-ouvertes.fr/hal-00510281/.

Langley, Pat et Herbert A. Simon (nov. 1995), « Applications of Machine Learning and Rule Induction », in : *Communications of the ACM* 38.11, p. 54-64, DOI : 10.1145/219717.219768.

Langley, Pat, Herbert A. Simon et al. (1987), *Scientific Discovery : Computational Explorations of the Creative Processes*, Cambridge, Mass : MIT Press.

Larribeau, Antoine (2019), « Du bidouilleur amateur à l'informaticien, apprendre le détournement : une étude de la socialisation au hacking informatique », in : *Sociologies pratiques* 38.1, p. 59-70, DOI : 10.3917/sopr.038.0059.

Lassègue, Jean et Giuseppe Longo (2012), « What Is Turing's Comparison between Mechanism and Writing Worth ? », in : *Conference on Computability in Europe*, Berlin : Springer, p. 450-461.

Le Blanc, Gilles (2005), *L'industrie Dans l'économie Française, 1978-2003 : Une Étude Comparée*, Institut de l'entreprise.

Le Blond, Jean-Marie [1936] (1970), *Logique et méthode chez Aristote : étude sur la recherche des principes dans la physique aristotélicienne*, 2e éd., Bibliothèque d'histoire de la philosophie, Paris : Vrin.

Lee, Edward A. (2017), *Plato and the Nerd : The Creative Partnership of Humans and Technology*, Cambridge, Mass : MIT Press.

Lefèvre, Wolfgang, Jürgen Renn et Urs Schoepflin, éd. (2012), *The Power of Images in Early Modern Science*, Birkhäuser.

Leibniz, Gottfried Wilhelm (1923 –), *Sämtliche Schriften und Briefe*, Preussischen Akademie der Wissenschaften, Darmstadt : Akademie-Verlag.

— [1875-1890] (1962), *Philosophische Schriften*, éd. établie par C. J. Gerhardt, réimpression, 7 vol., Hildesheim : Olms.

— (1903), *Opuscules et fragments inédits de Leibniz*, éd. établie par Louis Couturat, Paris : Felix Alcan.

— (1986), *Nouveaux essais sur l'entendement humain*, Paris : Flammarion.

Lennox, James G. (1982), « Teleology, Chance, and Aristotle's Theory of Spontaneous Generation », in : *Journal of the History of Philosophy* 20.3, p. 219-238.

— (1984), « Aristotle on Chance », in : *Archiv für Geschichte der Philosophie* 66.1, p. 52-60.

— (2011), « Aristotle on Norms of Inquiry », in : *HOPOS : The Journal of the International Society for the History of Philosophy of Science* 1.1, p. 23-46, doi : 10.1086/658482.

Lerner, Michel-Pierre (2005), « The Origin and Meaning of "World System" », in : *Journal for the History of Astronomy* 36.4, p. 407-441.

Leroi-Gourhan, André (1965), *Le Geste et La Parole*, 2 vol., Sciences d'aujourd'hui, Paris : Albin Michel.

Leroy, Xavier (2016), « Formally Verifying a Compiler : What Does It Mean, Exactly ? », in : *ICALP 2016, 43rd International Colloquium on Automata, Languages, and Programming*, Rome, url : https://drops.dagstuhl.de/portals/extern/index.php?semnr=16012 (visité le 22/10/2022).

Lessig, Lawrence (2000), « Code Is Law », in : *Harvard magazine* 1, p. 2000.

Lesuisse, Roland (fév. 2018), *Histoire d'une plateforme de commerce électronique : Le système global de distribution SABRE de 1960 à 2010*, London : ISTE Group.

Levitin, Dmitri (juin 2021), « Newton on the Rules of Philosophizing and Hypotheses : New Evidence, New Conclusions », in : *Isis* 112.2, p. 242-265, doi : 10.1086/714774.

Levy, Arnon et William Bechtel (avr. 2013), « Abstraction and the Organization of Mechanisms », in : *Philosophy of Science* 80.2, p. 241-261, doi : 10.1086/670300.

Li, Hang (2022), « Language Models : Past, Present, and Future », in : *Communications of the ACM* 65.7, p. 56-63.

Lipovača, Miran (2011), *Learn You a Haskell for Great Good !*, url : http://learnyouahaskell.com/ (visité le 12/05/2022).

Liskov, Barbara (1996), « A History of CLU », in : *History of Programming Languages—II*, 1re éd., New York : ACM Books, p. 511-598, doi : 10.1145/234286.1057828.

Littré, Émile (1873), *Dictionnaire de la langue française*, url : https://gallica.bnf.fr/ark:/12148/bpt6k5460034d (visité le 29/11/2021).

Lolli, Gabriele, Marco Panza et Giorgio Venturi, éd. (2015), *From Logic to Practice : Italian Studies in the Philosophy of Mathematics*, t. 308, Boston Studies in the Philosophy and History of Science, Cham : Springer, doi : 10.1007/978-3-319-10434-8.

Lonati, Violetta, Andrej Brodnik et al. (juill. 2022), « Characterizing the Nature of Programs for Educational Purposes », in : *Proceedings of the Conference*, t. 2, ITiCSE '22, New York : ACM Books, p. 572-573, doi : 10.1145/3502717.3532173.

Lonati, Violetta, Dario Malchiodi et al. (2015), « Is Coding the Way to Go ? », in : *Informatics in Schools. Curricula, Competences, and Competitions*, sous la dir. d'Andrej Brodnik et Jan Vahrenhold, Lecture Notes in Computer Science, Cham : Springer, p. 165-174, doi : 10.1007/978-3-319-25396-1_15.

Longuenesse, Béatrice (1993), *Kant et le pouvoir de juger : sensibilité et discursivité dans l'Analytique transcendantale de la Critique de la raison pure*, Epiméthée, Paris : Presses universitaires de France.

Lorenz, Hendrik et Benjamin Morison (oct. 2019), « Aristotle's Empiricist Theory of Doxastic Knowledge », in : *Phronesis* 64.4, p. 431-464, doi : 10.1163/15685284-12341975.

Lothon, Florent (2012), *Les spécifications agiles*, URL : https://agiliste.fr/les-specifications-agiles/ (visité le 16/10/2022).

Ludwig, Thomas (nov. 2018), « Reproducibility in Computational Climate Science », in : *SAS Workshop – Epistemic Opacity in Computer Simulation and Machine Learning*, HLRS - Stuttgart.

Ma, Hang et Sven Koenig (oct. 2017), « AI Buzzwords Explained : Multi-Agent Path Finding (MAPF) », in : *AI Matters* 3.3, p. 15-19, DOI : 10.1145/3137574.3137579.

MacCarthy, Bart L. et al. (2016), « Supply Chain Evolution – Theory, Concepts and Science », in : *International Journal of Operations and Production Management* 36.12, p. 1696-1718, DOI : 10.1108/IJOPM-02-2016-0080.

Mach, Ernst [1883] (1919), *The Science Of Mechanics*, trad. par Thomas J. McCormack, Chicago : The Open Court Publishing Co., URL : http://archive.org/details/scienceofmechani011018mbp (visité le 22/04/2023).

Machamer, Peter, Lindley Darden et Carl F. Craver (mars 2000), « Thinking about Mechanisms », in : *Philosophy of Science* 67.1, p. 1-25, DOI : 10.1086/392759.

MacIntyre, Alasdair C. [1981] (1985), *After Virtue : A Study in Moral Theory*, 2e éd., London : Duckworth.

MacKenzie, Donald (jan. 2004), *Mechanizing Proof : Computing, Risk, and Trust*, Cambridge, Mass : MIT Press.

Madaan, Aman et al. (mai 2023), *Self-Refine : Iterative Refinement with Self-Feedback*, DOI : 10.48550/arXiv.2303.17651.

Madelrieux, Stéphane (2016), *La philosophie de John Dewey : repères*, Repères philosophiques, Paris : Vrin.

Mäenpää, Petri (1993), « The Art of Analysis : Logic and History of Problem-Solving. », thèse de doctorat, Helsinki : Teknillinen Korkeakolou.

— (1997), « From Backward Reduction to Configurational Analysis », in : *Analysis and Synthesis in Mathematics. History and Philosophy*, sous la dir. de Marco Panza et Michael Otte, t. 196, Boston Studies in the Philosophy and History of Science, Dordrecht : Kluwer, p. 201-226.

Mäenpää, Petri et Jan Von Plato (1990), « The Logic of Euclidean Construction Procedures », in : *Acta Philosophica Fennica* 39.

Mahoney, Michael Sean (juin 2011), *Histories of Computing*, Harvard University Press.

Mancosu, Paolo, éd. (juin 2008), *The Philosophy of Mathematical Practice*, Oxford : Oxford University Press.

Manna, Zohar et Amir Pnueli (1992), *The Temporal Logic of Reactive and Concurrent Systems - Specification*, t. 1, New York : Springer.

Manovich, Lev (fév. 2002), *The Language of New Media*, Cambridge, Mass : MIT Press.

Marchand, Stéphane (nov. 2015), « Sextus Empiricus, scepticisme et philosophie de la vie quotidienne », in : *Philosophie antique. Problèmes, Renaissances, Usages* 15, p. 91-119, doi : 10.4000/philosant.366.

Maronne, Sébastien (mai 2017), « Qu'est-ce qu'une action en mathématiques ? », in : *Workshop « Histoire et philosophie des mathématiques »*, Faculdad de Ciencias, UNAM, Mexico, url : https://www.youtube.com/watch?v=Lsf-23M5TJA&t=12s (visité le 10/01/2023).

Martin, Christian, éd. (2018), *Language, Form(s) of Life, and Logic*, Berlin : Walter de Gruyter.

Martin, Robert C., éd. (2009), *Clean Code : A Handbook of Agile Software Craftsmanship*, Upper Saddle River, NJ : Prentice Hall.

Martin-Löf, Per (1982), « Constructive Mathematics And Computer Programming », in : *Logic, Methodology and Philosophy of Science VI*, sous la dir. de J. J. Cohen et al., Studies in Logic and the Foundations of Mathematics 104, Amsterdam : North-Holland, p. 153-174.

— (1984), *Intuitionistic Type Theory (Padua 1980)*, t. 9, Napoli : Bibliopolis.

— (1987), « Truth of a Proposition, Evidence of a Judgement, Validity of a Proof », in : *Synthese* 73.3, p. 407-420.

— (1991), « A Path from Logics to Metaphysics », in : *Atti del Congresso "Nuovi problemi della logica e della filosofia della scienza"*, sous la dir. de Giovanna Corsi, Giovanni Sambin et Maria Luisa Dalla Chiara, t. 2, Bologna : Cooperativa Libraria Universitaria Editrice Bologna, p. 141-149.

— (1994), « Analytic and Synthetic Judgements in Type Theory », in : *Kant and Contemporary Epistemology*, sous la dir. de Paolo Parrini, Dordrecht : Springer, p. 87-99, doi : 10.1007/978-94-011-0834-8_5.

— (1995), « Verificationism Then and Now », in : *The Foundational Debate : Complexity and Constructivity in Mathematics and Physics*, sous la dir. de

W. DePauli-Schimanovic, E. Köhler et F. Stadler, Dordrecht : Kluwer, p. 187-196.

— [1983] (1996), « On the Meanings of the Logical Constants and the Justifications of the Logical Laws (Siena 1985) », in : *Nordic journal of philosophical logic* 1.1, p. 11-60.

Martini, Simone (nov. 2021), « Programmer, c'est un peu comme jouer aux LEGO : dimensions non-verbales de la programmation », in : *Séminiaire UNA Europa*, Université Paris 1 Panthéon-Sorbonne.

Marx, Karl [1867] (2006), *Le capital : critique de l'économie politique. Livre I*, trad. par Jean-Pierre Lefebvre, 2e éd., Quadrige, Paris : Presses universitaires de France.

Mashkoor, Atif, Felix Kossak et Alexander Egyed (2018), « Evaluating the Suitability of State-Based Formal Methods for Industrial Deployment », in : *Software : Practice and Experience* 48.12, p. 2350-2379.

Mazzoni, Jacopo [1597] (2010), *In universam Platonis et Aristotelis philosophiam Praeludia, sive De comparatione Platonis et Aristotelis*, éd. établie par Sara Matteoli et Anna De Pace, Storie e testi 19, Napoli : M. D'Auria.

McBreen, Pete (août 2001), *Software Craftsmanship : The New Imperative*, Boston : Addison-Wesley.

McCulloch, Warren S. et Walter Pitts (1943), « A Logical Calculus of the Ideas Immanent in Nervous Activity », in : *The bulletin of mathematical biophysics* 5.4, p. 115-133.

McMullin, Ernan (1978), « Structural Explanation », in : *American Philosophical Quarterly* 15.2, p. 139-147.

— (2001), « The Impact of Newton's Principia on the Philosophy of Science », in : *Philosophy of Science* 68.3, p. 279-310.

Meijers, Anthonie W. M. et Peter A. Kroes (2013), « Extending the Scope of the Theory of Knowledge », in : *Norms in Technology*, sous la dir. de Marc J. de Vries, Sven Ove Hansson et Anthonie W.M. Meijers, Philosophy of Engineering and Technology, Dordrecht : Springer, p. 15-34, doi : 10.1007/978-94-007-5243-6_2.

Mélès, Baptiste (2022), « Tous les fichiers sont des programmes - et non l'inverse », in : *La Gazette des Mathématiciens*.

Mérigoux, Denis (mars 2021), « Catala : Un Langage de Programmation Pour La Loi Socio-Fiscale », in : *Séminaire Code Sources*.

Merriam-Webster (1936), *Webster's Collegiate Dictionary*, 5th, Springfield, Mass. : G. & C. Merriam Co., URL : http://archive.org/details/websterscollegia00spri (visité le 13/11/2021).

Meyer, Bertrand (2008), « Eiffel as a Framework for Verification », in : *Verified Software : Theories, Tools, Experiments*, sous la dir. de Bertrand Meyer et Jim Woodcock, Lecture Notes in Computer Science, Zurich : Springer, p. 301-307, DOI : 10.1007/978-3-540-69149-5_32.

Michalewicz, Zbigniew et David B. Fogel [2000] (2004), *How to Solve It : Modern Heuristics*, 2e éd., Berlin : Springer.

Michel, Christian (2008), « La peinture peut-elle être réduite en art ? », in : *Réduire en art : la technologie de la Renaissance aux Lumières*, sous la dir. de Hélène Vérin et Pascal Dubourg Glatigny, Paris : Éditions de la Maison des sciences de l'homme, p. 106-118.

Michelfelder, Diane P. et Neelke Doorn, éd. (déc. 2020), *The Routledge Handbook of the Philosophy of Engineering*, New York : Routledge, DOI : 10.4324/9781315276502.

Mignucci, Mario (1981), « Hos Epi to Polu et Nécessaire Dans La Conception Aristotélicienne de La Science », in : *Aristotle on Science : The Posterior Analytics*, sous la dir. d'Enrico Berti, Università Di Padova, Padova : Editrice Antenore, p. 173-203.

Miller, George A., Eugene Galanter et Karl H. Pribram [1960] (2013), *Plans and the Structure of Behavior*, Martino Fine Books.

Milner, Robin (1993), « Turing Award Lecture : Elements of Interaction », in : *Communications of the ACM* 36.1, p. 78-90.

— (1999), *Communicating and Mobile Systems : The [Pi]-Calculus*, Cambridge : Cambridge University Press.

Minsky, Marvin Lee (1967), « Why Programming Is a Good Medium for Expressing Poorly Understood and Sloppily Formulated Ideas », in : *Design and Planning II-computers in Design and Communication*, sous la dir. de Martin Krampen et Peter Seitz, New York : Hastings House Publishers, p. 120-125.

Mohr-Schroeder, Margaret J., Maureen Cavalcanti et Kayla Blyman (jan. 2015), « STEM Education : Understanding the Changing Landscape », in : *A Practice-Based Model of STEM Teaching*, sous la dir. d'Alpalsan Sahin, Brill, p. 3-14.

Mokyr, Joel (2016), *A Culture of Growth : The Origins of the Modern Economy*, Princeton : Princeton University Press.

Moor, James H. (1978), « Three Myths of Computer Science », in : *The British Journal for the Philosophy of Science* 29.3, p. 213-222.

Morgan, Carleigh (août 2019), « Calculated Error : Glitch Art, Compression Artefacts, and Digital Materiality », in : *A Peer-Reviewed Journal About* 8.1, p. 204-217, doi : 10.7146/aprja.v8i1.115426.

Morgan, Mary S. et Margaret Morrison, éd. (1999), *Models as Mediators : Perspectives on Natural and Social Science*, Ideas in Context 52, Cambridge : Cambridge University Press.

Moschovakis, Yiannis N. (2001), « What Is an Algorithm ? », in : *Mathematics Unlimited — 2001 and Beyond*, sous la dir. de Björn Engquist et Wilfried Schmid, Berlin : Springer, p. 919-936, doi : 10.1007/978-3-642-56478-9_46.

Moss, Jessica (jan. 2011), « Virtue Makes the Goal Right : Virtue and Phronesis in Aristotle's Ethics », in : *Phronesis* 56.3, p. 204-261, doi : 10.1163/156852811X575907.

— (2014), « Right Reason in Plato and Aristotle : On the Meaning of Logos », in : *Phronesis* 59.3, p. 181-230.

Mukhtar, Maryam I. et Bashir S. Galadanci (2018), « Automatic Code Generation from UML Diagrams : The State-of-the-Art », in : *Science World Journal* 13.4, p. 47-60, doi : 10.4314/swj.v13i4.

Müller, Jozef (avr. 2018), « Practical and Productive Thinking in Aristotle », in : *Phronesis* 63.2, p. 148-175, doi : 10.1163/15685284-12341345.

Muro, Mark, Sifan Liu et Siddharth Kulkarni (nov. 2017), *Digitalization and the American Workforce*, rapp. tech., Washington DC.

Naibo, Alberto (2015), « Constructibility and Geometry », in : *From Logic to Practice : Italian Studies in the Philosophy of Mathematics*, sous la dir. de Gabriele Lolli, Marco Panza et Giorgio Venturi, Boston Studies in the Philosophy and History of Science 308, Cham : Springer, p. 123-161, doi : 10.1007/978-3-319-10434-8.

— (juin 2022), *Les Enjeux Épistémologiques Des Algorithmes*, Paris.

Naibo, Alberto et Marco Panza (oct. 2021), « Formalizing the Logic and Proofs of Euclid's Elements, Book I », in : *Conférence : L'intuitionnisme Entre Philosophie, Mathématique et Logique : Mutations et Histoire Longue*, Paris.

Naibo, Alberto et Thomas Seiller (jan. 2022), « A Geometric Theory of Algorithms », in : *Séminaire d'histoire et Philosophie Des Mathématiques - SPHERE*, Paris.

Napoletani, Domenico, Marco Panza et Daniele C. Struppa (fév. 2011), « Agnostic Science. Towards a Philosophy of Data Analysis », in : *Foundations of Science* 16.1, p. 1-20, doi : 10.1007/s10699-010-9186-7.

Naur, Peter (mai 1985), « Programming as Theory Building », in : *Microprocessing and Microprogramming* 15.5, p. 253-261, doi : 10.1016/0165-6074(85)90032-8.

Neurath, Otto (1973), *Empiricism and Sociology*, sous la dir. de Marie Neurath et Robert S. Cohen, Dordrecht : Springer, doi : 10.1007/978-94-010-2525-6.

Newell, Allen, John C. Shaw et Herbert A. Simon (1959), *Report on a General Problem Solving Program*, rapp. tech. 1584, Pittsburgh.

Newton, Isaac (1959), *The Correspondence of Isaac Newton*, sous la dir. d'Alfred Rupert Hall et al., Cambridge : Cambridge University Press.

Nicał, Aleksander et Wojciech Wodyński (déc. 2016), « Enhancing Facility Management through BIM 6D », in : *Procedia Engineering* 164, p. 299-306, doi : 10.1016/j.proeng.2016.11.623.

Niiniluoto, Ilkka (jan. 1993), « The Aim and Structure of Applied Research », in : *Erkenntnis* 38.1, p. 1-21, doi : 10.1007/BF01129020.

Nordström, Bengt, Kent Petersson et Jan M. Smith (1990), *Programming in Martin-Löf's Type Theory*, Oxford : Oxford University Press, url : www.cs.chalmers.se/Cs/Research/Logic.

Nuseibeh, Bashar et Pamela Zave, éd. (2010), *Software Requirements and Design : The Work of Michael Jackson*, Chatham, NJ : Good Friends Publishing.

Nussbaum, Martha Craven [1986] (2001), *The Fragility of Goodness : Luck and Ethics in Greek Tragedy and Philosophy*, 2e éd., Cambridge Paperback Library, Cambridge : Cambridge University Press.

Oberman, Heiko A. (oct. 2003), « Luther and the Via Moderna : The Philosophical Backdrop of the Reformation Breakthrough », in : *The Journal of Ecclesiastical History* 54.4, p. 641-670, doi : 10.1017/S0022046903008005.

Ong, Walter Jackson (1958), *Ramus, Method and the Decay of Dialogue : From the Art of Discourse to the Art of Reason*, Cambridge, Mass : Harvard University Press.

Orr, Julian E. [1996] (2016), *Talking about Machines : An Ethnography of a Modern Job*, Ithaca : Cornell University Press.

Osta Vélez, Matías (2020), « Inference and the Structure of Concepts », thèse de doctorat, Paris - München : Université Paris 1, Ludwig-Maximilians Universität.

Pagès, Gaspard et Catherine Verna (nov. 2022), « L'invention de l'industrie antique et médiévale », in : *Artefact. Techniques, histoire et sciences humaines* 17, p. 243-264, doi : 10.4000/artefact.13309.

Palmerino, Carla Rita et al., éd. (2004), *The Reception of the Galilean Science of Motion in Seventeenth-Century Europe*, Boston Studies in the Philosophy of Science 239, Dordrecht : Springer, doi : 10.1007/978-1-4020-2455-9.

Papert, Seymour (mai 1980), *Mindstorms : Children, Computers and Powerful Ideas*, Brighton : Prentice Hall / Harvester Wheatsheaf.

Parent, Monelle (2004), « La technoscience, un défi pour la philosophie », in : *revue Phares* 5.

Pautet, Sébastien (août 2018), « Les ingénieurs à l'époque moderne », in : *L'Europe des sciences et des techniques : Un dialogue des savoirs, XVe-XVIIIe siècle*, sous la dir. de Liliane Hilaire-Pérez, Fabien Simon et Marie Thébaud-Sorger, Histoire, Rennes : Presses universitaires de Rennes, p. 111-121, doi : 10.4000/books.pur.45892.

Peaucelle, Jean-louis (2012), « Un « éléphant blanc » en pleine Révolution Française : les grandes tables de logarithmes de Prony comme substitut au cadastre ! », in : *Annales des Mines - Gerer et comprendre* N° 107.1, p. 74-86.

Pedersen, Olaf (1961), « Theorica - A Study in Language and Civilization », in : *Classica et Mediaevalia* 22, p. 151-175.

Pégny, Maël (à paraître), *L'éthique Des Algorithmes*, Paris : Vrin.

Pennington, Nancy et Beatrice Grabowski (1990), « The Tasks of Programming », in : *Psychology of Programming*, sous la dir. de J.-M. Hoc et al., Academic Press, p. 45-62.

Pereira, Henrique M. et Laetitia M. Navarro, éd. (2015), *Rewilding European Landscapes*, Cham : Springer, doi : 10.1007/978-3-319-12039-3.

Pérez-Ramos, Antonio (1988), *Francis Bacon's Idea of Science and the Maker's Knowledge Tradition*, Oxford : Clarendon Press.

Perlis, Alan J. (sept. 1982), « Epigrams on Programming », in : *SIGPLAN Notices* 17.9, p. 7-13, doi : 10.1145/947955.1083808.

Petzold, Charles (oct. 2000), *Code : The Hidden Language of Computer Hardware and Software*, Microsoft Press.

Philopon, Jean (1897), *Ioannis Philoponi in Aristotelis de Anima libros commentaria*, éd. établie par Michael Hayduck, Commentaria in Aristotelem Graeca XV, Berlin : G. Reimeri.

Piantadosi, Steven (mars 2023), *Modern Language Models Refute Chomsky's Approach to Language*.

Piccinini, Gualtiero (2015), *Physical Computation : A Mechanistic Account*, Oxford : Oxford University Press.

Pierce, Benjamin C. et al. (2023), *Logical Foundations*, Software Foundations, url : https://softwarefoundations.cis.upenn.edu/lf-current/index.html (visité le 16/05/2023).

Place, Ullin T. (1999), « Ryle's Behaviorism », in : *Handbook of Behaviorism*, Elsevier, p. 361-398.

Platon (1993), *La République*, trad. par Pierre Pachet, Folio Essais, Paris : Gallimard.

Pline l'Ancien (1972), *Histoire Naturelle*, éd. établie et trad. par Henri Le Bonniec, t. XVIII, Paris : Les Belles Lettres.

Pollock, Neil et Robin Williams (2008), *Software and Organisations : The Biography of the Enterprise-Wide System or How SAP Conquered the World*, Routledge.

Pólya, George [1945] (1990), *How to Solve It : A New Aspect of Mathematical Method*, 2nd, London : Penguin Books.

Popplow, Marcus (2007), « Setting the World Machine in Motion : The Meaning of Machina Mundi in the Middle Ages and the Early Modern Period », in : *Mechanics and Cosmology in the Medieval and Early Modern Period*, sous la dir. de Massimo Bucciantini, Michele Camerota et Sophie Roux, Biblioteca Di Nuncius LXIV, Firenze : L. S. Olschki, p. 45-70.

Posy, Carl (1998), « Brouwer versus Hilbert : 1907–1928 », in : *Science in Context* 11.2, p. 291-325, doi : 10.1017/S0269889700003021.

Pratt, Vaughan (1976), *Semantical Considerations on Floyd-Hoare Logic*, Technical Report LCS-TR-168, Cambridge, Mass.

— (1990), « Action Logic and Pure Induction », in : *European Workshop on Logics in Artificial Intelligence*, Berlin : Springer, p. 97-120.

Preidel, Cornelius et André Borrmann (2018), « BIM-Based Code Compliance Checking », in : *Building Information Modeling – Technology Foundations and Industry Practice*, sous la dir. d'André Borrmann et al., Cham : Springer, p. 367-381, doi : 10.1007/978-3-319-92862-3_22.

Presti, Roberto Lo (jan. 2012), « La notion d'automaton dans les textes médicaux (Hippocrate et Galien) et la Physique d'Aristote », in : *Les Études Classiques* 80.1-2.

Priestley, Mark (2011), *A Science of Operations : Machines, Logic and the Invention of Programming*, History of Computing, London : Springer.

— (sept. 2017), « AI and the Origins of the Functional Programming Language Style », in : *Minds and Machines* 27.3, p. 449-472, doi : 10.1007/s11023-017-9432-7.

— (2021), « Logic, Code, and the History of Programming », in : *IEEE Annals of the History of Computing* 43.4, p. 92-96.

Primiero, Giuseppe (jan. 2020), *On the Foundations of Computing*, New York : Oxford University Press.

Proclus (1970), *A Commentary on the First Book of Euclid's Elements*, trad. par Glenn R. Morrow, Princeton : Princeton University Press.

PROGRAMme (à paraître), *What Is a Computer Program ?*, url : https://wiki.program-me.org/index.php/Book_I:Outline#Chapter_1:_What_is_a_computer_program.3F (visité le 13/08/2022).

Ptolémée (1984), *Almagest*, éd. établie par Gerald J. Toomer, London : Duckworth.

Quilty-Harper, Conrad (mars 2023), « $335,000 Pay for 'AI Whisperer' Jobs Appears in Red-Hot Market », in : *Bloomberg.com*, url : https://news.bloomberglaw.com/daily-labor-report/335-000-pay-for-ai-whisperer-jobs-appears-in-red-hot-market (visité le 04/04/2023).

Railton, Peter (1981), « Probability, Explanation, and Information », in : *Synthese*, p. 233-256.

Regenwetter, Lyle, Amin Heyrani Nobari et Faez Ahmed (mars 2022), « Deep Generative Models in Engineering Design : A Review », in : *Journal of Mechanical Design* 144.7, doi : 10.1115/1.4053859.

Rescher, Nicholas (1966), *The Logic of Commands*, Monographs in Modern Logic, London : Routledge and K. Paul.

Ritschl, Otto (1906), *System und systematische Methode in der Geschichte des wissenschaftlichen Sprachgebrauchs und der philosophischen Methodologie*, Bonn : Carl Georgi, URL : http://archive.org/details/ORitschelSystemUndSystematischeMethode (visité le 11/02/2023).

Ritter, Jim (jan. 2010), « Translating Rational-Practice Texts », in : *Writings of Early Scholars in the Ancient Near East, Egypt, Rome, and Greece*, sous la dir. d'Annette Imhausen et Tanja Pommerening, Beiträge Zur Altertumskunde, Berlin : De Gruyter, DOI : 10.1515/9783110229936.

Ritter, Joachim et al. (1980), « Methode », in : *Historisches Wörterbuch der Philosophie* 5, sous la dir. de Joachim Ritter et al., p. 1304-1332.

Robert, Paul et Alain Rey (1986), *Dictionnaire Alphabétique et Analogique de La Langue Française*, 2e éd., Paris : Le Robert.

Roberts, Lissa, Simon Schaffer et Peter Robert Dear, éd. (2007), *The Mindful Hand : Inquiry and Invention from the Late Renaissance to Early Industrialisation*, History of Science and Scholarship in the Netherlands 9, Amsterdam : Koninklijke Nederlandse Akademie van Wetenschappen.

Robinson, Richard (1963), *Definition*, Oxford University Press.

Rodier, Georges (1926), *Etudes de philosophie grecque*, Bibliothèque d'histoire de la philosophie, Paris : Vrin.

Rohrhuber, Julian (fév. 2018), « Algorithmic Music and the Philosophy of Time », in : *The Oxford Handbook of Algorithmic Music*, sous la dir. d'Alex McLean et Roger T. Dean, Oxford : Oxford University Press, DOI : 10.5281/zenodo.2596675.

Rossi, Paolo [1962] (1996), *Les philosophes et les machines, 1400-1700*, trad. par Patrick Vighetti, Science, histoire et société, Paris : Presses universitaires de France.

Roux, Sophie (2004), « Cartesian Mechanics », in : *The Reception of the Galilean Science of Motion in Seventeenth-Century Europe*, sous la dir. de Carla Rita Palmerino et al., Boston Studies in the Philosophy of Science 239, Dordrecht : Springer, p. 25-66, DOI : 10.1007/978-1-4020-2455-9.

— (2013), « Quelles machines pour quels animaux ? Jacques Rohault, Claude Perrault, Giovanni Alfonso Borelli », in : *L'automate : modèle, métaphore, machine, merveille. Colloque international de Grenoble, 19-21 mars 2009*, sous la dir. d'Aurélia Gaillard et al., Mirabilia, Pessac : Presses universitaires de Bordeaux, p. 69-113.

— (juill. 2017), « From the Mechanical Philosophy to Early Modern Mechanisms », in : *The Routledge Handbook of Mechanisms and Mechanical Philosophy*, sous la dir. de Stuart Glennan et Phyllis Illari, Taylor & Francis, p. 26-45.

Roux, Sophie et E. Festa (2001), « Le para phusin et l'imitation de la nature dans quelques commentaires du prologue des Questions mécaniques », in : *Largo campo di filosofare : Eurosymposium Galileo 2001*, sous la dir. de José Montesinos, Carlos Solís et Jean-Yves Dupont, La Orotava : Fundación Canaria Orotava de Historia de la Ciencia, p. 237-253.

Ryle, Gilbert (1945), « Knowing How and Knowing That : The Presidential Address », in : *Proceedings of the Aristotelian Society* 46, p. 1-16.

— [1949] (2002), *The Concept of Mind*, Chicago : University of Chicago Press.

— [1971] (2009), *Collected Essays 1929 - 1968*, Abingdon : Routledge.

Samuel, Arthur L. (1959), « Some Studies in Machine Learning Using the Game of Checkers », in : *IBM Journal* 3.3, p. 535-554.

Sarri, Anna et al. (2022), *Building Effective Governance Frameworks for the Implementation of National Cybersecurity Strategies*. Rapp. tech., Luxembourg, url : https://data.europa.eu/doi/10.2824/850466 (visité le 07/08/2023).

Schatzberg, Eric (2018), *Technology : Critical History of a Concept*, Chicago : University of Chicago Press.

Schepers, H. (1971), « Heuristik, Heuristisch », in : *Historisches Wörterbuch der Philosophie* 3, sous la dir. de Joachim Ritter et al., p. 1115-1120.

Schick, Timo et al. (fév. 2023), *Toolformer : Language Models Can Teach Themselves to Use Tools*, doi : 10.48550/arXiv.2302.04761.

Schlanger, Nathan (jan. 2023), *L'invention de la technologie : Une histoire intellectuelle avec André Leroi-Gourhan*, Paris : Presses universitaires de France.

Schmidt, Kjeld (2011), *Cooperative Work and Coordinative Practices*, Computer Supported Cooperative Work, London : Springer, doi : 10.1007/978-1-84800-068-1.

Schmidt-Biggemann, Wilhelm (1983), *Topica universalis : eine Modellgeschichte humanistischer und barocker Wissenschaft*, Paradeigmata 1, Hamburg : Meiner.

Scholtes, Maike et al. (2021), « 6-Layer Model for a Structured Description and Categorization of Urban Traffic and Environment », in : *IEEE Access* 9, p. 59131-59147, DOI : 10.1109/ACCESS.2021.3072739.

Schrape, Jan-Felix (2019), « Open-Source Projects as Incubators of Innovation : From Niche Phenomenon to Integral Part of the Industry », in : *Convergence* 25.3, p. 409-427.

Searle, John R. (1980), « Minds, Brains, and Programs », in : *The Behavioral and Brain Sciences*, p. 42.

Sebestik, Jan (déc. 2007), « Les commencements de la technologie. Postface/préface », in : *Documents pour l'histoire des techniques. Nouvelle série* 14, p. 123-133.

Séris, Jean-Pierre (jan. 1994a), *La Technique*, Paris : Presses universitaires de France.

— (1994b), *Qu'est-ce que la division du travail ?*, Pré-textes 6, Paris : Vrin.

Shank, Michael H. (2007), « Mechanical Thinking in European Astronomy (13th-15th Centuries) », in : *Mechanics and Cosmology in the Medieval and Early Modern Period*, sous la dir. de Massimo Bucciantini, Michele Camerota et Sophie Roux, Biblioteca Di Nuncius LXIV, Firenze : L. S. Olschki, p. 3-28.

Shuermans, Stijn et Christina Voskoglou (2019), *Global Developer Population 2019*, rapp. tech., Manchester, URL : https://www.slashdata.co/free-resources? (visité le 29/07/2023).

Sidoli, Nathan (juill. 2018), « Uses of Construction in Problems and Theorems in Euclid's Elements I-VI », in : *Archive for History of Exact Sciences* 72.4, p. 403-452, DOI : 10.1007/s00407-018-0212-4.

Sidoli, Nathan et Ken Saito (2009), « The Role of Geometrical Construction in Theodosius's Spherics », in : *Archive for History of Exact Sciences* 63.6, p. 581-609.

Sieg, Wilfried (juin 2006), « Gödel on Computability », in : *Philosophia Mathematica* 14.2, p. 189-207, DOI : 10.1093/philmat/nkj005.

— (2009), « On Computability », in : *Philosophy of mathematics* 4, p. 549-630.

Simon, Herbert A. (1972), « Complexity and the Representation of Patterned Sequences of Symbols », in : *Psychological Review* 79.5, p. 369-382, DOI : 10.1037/h0033118.

— (1976a), « From Substantive to Procedural Rationality », in : *25 Years of Economic Theory : Retrospect and Prospect*, sous la dir. de T. J. Kastelein et al., Boston : Springer, p. 65-86, doi : 10.1007/978-1-4613-4367-7_6.
— (1976b), « How Complex Are Complex Systems ? », in : *PSA : Proceedings of the Biennial Meeting of the Philosophy of Science Association* 1976.2, p. 507-522.
— [1967] (1977), « The Logic of Heuristic Decision-Making », in : *Models of Discovery : And Other Topics in the Methods of Science*, Boston Studies in the Philosophy of Science v. 54, Dordrecht : D. Reidel Pub. Co.
— (1978a), « On the Forms of Mental Representations », in : *Perception and Cognition. Issues in the Foundations of Psychology*, sous la dir. de C. Wade Savage, Minnesota Studies in the Philosophy of Science IX, Minneapolis : University of Minnesota Press, p. 3-18.
— (1978b), « Rationality as Process and as Product of Thought », in : *The American Economic Review* 68.2, p. 1-16.
— (1980), « Cognitive Science : The Newest Science of the Artificial », in : *Cognitive science* 4.1, p. 33-46.
— (1982), « Unity of the Arts and Sciences : The Psychology of Thought and Discovery », in : *Bulletin of the American Academy of Arts and Sciences* 35.6, p. 26-53, doi : 10.2307/3823595.
— (jan. 1983), « Why Should Machines Learn ? », in : *Machine Learning*, sous la dir. de Ryszard S. Michalski, Jaime G. Carbonell et Tom M. Mitchell, San Francisco (CA) : Morgan Kaufmann, p. 25-37, doi : 10.1016/B978-0-08-051054-5.50006-6.
— (fév. 1984), « Expert and Novice Problem Solving in Science and Mathematics », in : *Autour de l'oeuvre de H. A. Simon : Le Paradigme S.T.I.* Université Paul Valéry, Montpellier.
— (1986), « Whether Software Engineering Needs to Be Artificially Intelligent », in : *IEEE Transactions on Software Engineering* 7, p. 726-732.
— (1987), « Two Heads Are Better than One : The Collaboration between AI and OR », in : *Interfaces* 17.4, p. 8-15.
— (juin 1994), « Computers in Design Education », in : *An International Workshop on the Future Directions of Computer Aided Engineering – Tribute of Steven Fenves*.

Simon, Herbert A. (oct. 1996a), « Computational Theories of Cognition », in : *The Philosophy of Psychology*, sous la dir. de William O'Donohue et Richard F. Kitchener, SAGE.

— [1991] (1996b), *Models of My Life*, Cambridge, Mass : MIT Press.

— [1969] (1996c), *The Sciences of the Artificial*, 3e éd., Cambridge, Mass : MIT Press.

— [1947] (1997), *Administrative Behavior : A Study of Decision-Making Processes in Administrative Organizations*, 4e éd., New York : Free Press.

Simon, Herbert A., Pat Langley et Gary L. Bradshaw (1981), « Scientific Discovery as Problem Solving », in : *Synthese* 47, p. 1-27.

Simon, Herbert A. et Allen Newell (1971), « Human Problem Solving : The State of the Theory in 1970 », in : *American Psychologist* 26.2, p. 145-159, doi : 10.1037/h0030806.

— (1972), *Human Problem Solving*, Englewood Cliffs, N.J : Prentice Hall.

Sipser, Michael (1997), *Introduction to the Theory of Computation*, Boston : PWS.

Skinner, Burrhus F. (1938), *The Behavior of Organisms*, Century Psychology Series, New York : Appleton-Century-Crofts.

Smith, Brian Cantwell (jan. 1985), « The Limits of Correctness », in : *ACM SIGCAS Computers and Society* 14,15.1,2,3,4, p. 18-26, doi : 10.1145/379486.379512.

Soh, Christina, Sia Siew Kien et Joanne Tay-Yap (avr. 2000), « Enterprise Resource Planning : Cultural Fits and Misfits : Is ERP a Universal Solution ? », in : *Communications of the ACM* 43.4, p. 47-51, doi : 10.1145/332051.332070.

Soh, Christina et Siew Kien Sia (déc. 2004), « An Institutional Perspective on Sources of ERP Package–Organisation Misalignments », in : *The Journal of Strategic Information Systems* 13.4, p. 375-397, doi : 10.1016/j.jsis.2004.11.001.

Sperber, Dan et Hugo Mercier (avr. 2017), *The Enigma of Reason : A New Theory of Human Understanding*, Penguin.

Steiner, Pierre (2010), « Philosophie, technologie et cognition. Etats des lieux et perspectives », in : *Intellectica* 53.1, p. 7-40, doi : 10.3406/intel.2010.1176.

Stephanou, Henri (2018), « Epistemic Opacity and Understanding », in : *SAS Workshop – Epistemic Opacity in Computer Simulation and Machine Learning*, HLRS - Stuttgart.

— (2022), « « Cette bonne vieille intelligence artificielle » : désuétude et survivance de l'hypothèse computationnelle de l'esprit », in : *Philosophia Scientiæ* 26.1, p. 73-91.

Stephenson, Amanda (juin 2023), « Mining Companies Betting on Autonomous Technology to Make Dangerous Jobs Safer | CBC News », in : *CBC News*, url : https://www.cbc.ca/news/canada/calgary/mining-companies-betting-on-autonomous-technology-to-make-dangerous-jobs-safer-1.6888075 (visité le 12/07/2023).

Stern, Leah Jacob (avr. 1974), « Empirical Concepts as Rules in the Critique of Pure Reason », in : *Akten des 4. Internationalen Kant-Kongresses*, sous la dir. de Gerhard Funke, t. II.1, Mainz : W. de Gruyter, p. 152-165.

Stock, Brian (1975), « Experience, Praxis, Work, and Planning in Bernard of Clairvaux : Observations on the Sermones in Cantica », in : *The Cultural Context of Medieval Learning*, sous la dir. de John Emery Murdoch et Edith d. Sylla, Dordrecht : D. Reidel Pub. Co, p. 219-268, url : https://archive.org/details/culturalcontexto0000inte/page/436/mode/2up.

Stumpe, Joe (mai 2018), *Symbiosis : Why CFD and Wind Tunnels Need Each Other*, url : https://aerospaceamerica.aiaa.org/features/symbiosis-why-cfd-and-wind-tunnels-need-each-other/ (visité le 21/07/2023).

Suchman, Lucy A. (nov. 1987), *Plans and Situated Actions : The Problem of Human-Machine Communication*, Cambridge University Press.

Sundholm, Göran (1993), « Questions of Proof », in : *Manuscrito* 16.2, p. 47-70.

— (1997), « Implicit Epistemic Aspects of Constructive Logic », in : *Journal of Logic, Language and Information* 6.2, p. 191-212.

— (1998), « Inference versus Consequence », in : *The Logica Yearbook*, sous la dir. de Timothy Childers, Praha : Filosofia. Institute of Philosophy, Academy of Sciences of the Czech Republic, p. 26-36.

Swade, Doron (jan. 2011), « Inventing the User : EDSAC in Context », in : *The Computer Journal* 54.1, p. 143-147, doi : 10.1093/comjnl/bxp116.

Symons, John et Ramón Alvarado (2019), « Epistemic Entitlements and the Practice of Computer Simulation », in : *Minds and Machines* 29.1, p. 37-60.

Tarski, Alfred (1959), « What Is Elementary Geometry ? », in : *The Axiomatic Method*, sous la dir. de Leon Henkin, Patrick Suppes et Alfred Tarksi, Studies in Logic and the Foundations of Mathematics 27, Elsevier, p. 16-29, doi : 10.1016/S0049-237X(09)70017-5.

Tedre, Matti (déc. 2014), *The Science of Computing : Shaping a Discipline*, Chapman and Hall/CRC, doi : 10.1201/b17764, (visité le 28/04/2019).

Tedre, Matti et Peter J. Denning (2016), « The Long Quest for Computational Thinking », in : *Proceedings of the 16th Koli Calling International Conference on Computing Education Research*, Koli Calling '16, New York : ACM Books, p. 120-129, doi : 10.1145/2999541.2999542.

Terwilliger, Thomas C. et al. (mai 2023), *AlphaFold Predictions Are Valuable Hypotheses, and Accelerate but Do Not Replace Experimental Structure Determination*, doi : 10.1101/2022.11.21.517405.

Thomas, David et Andrew Hunt [1999] (2020), *The Pragmatic Programmer*, 20th anniversary ed., Boston : Addison-Wesley.

Thomas, Ivo (1965), « The Later History of the Pons Asinorum », in : *Contributions to Logic and Methodology in Honor of J. M. Bochenski*, sous la dir. d'Anna-Theresa Tymieniecka, Amsterdam : North-Holland, p. 142-150, doi : 10.1016/B978-1-4832-3159-4.50015-0.

Thomas d'Aquin (1970), *Quaestiones disputatae de veritate*, 3 vol., Sancti Thomae de Aquino Opera omnia 22, Roma : ad. Sanctae Sabinae.

— (1984), *Somme théologique*, éd. établie par Albert Raulin, trad. par Aimon-Marie Roguet, 4 vol., Paris : Cerf.

— (1992), *Super Boetium de Trinitate Expositio libri Boetii de Ebdomadibus*, Roma : Commissio Leonina ; Paris : Ed. du Cerf.

— (2019), *Commentaire Des Sentences de Pierre Lombard*, éd. établie par Serge Pronovost, éd. numérique, url : http://docteurangelique.free.fr/bibliotheque/sommes/SENTENCES1.htm#_Toc516173971 (visité le 24/09/2023).

Thummadi, Babu et Kalle Lyytinen (juill. 2020), « How Much Method-in-Use Matters ? A Case Study of Agile and Waterfall Software Projects and Their Design Routine Variation », in : *Journal of the Association for Information Systems* 21.4, doi : 10.17705/1jais.00623.

Tolman, Edward C. (1948), « Cognitive Maps in Rats and Men. », in : *Psychological review* 55.4, p. 189.

Torvalds, Linus (avr. 2002), *Re : [PATCH] Remove Bitkeeper Documentation from Linux Tree*, URL : https://lwn.net/2002/0425/a/ideology-sucks.php3 (visité le 16/01/2023).

Traub, J. F. (juin 1981), « Quo Vadimus : Computer Science in a Decade », in : *Communications of the ACM* 24.6, p. 351-369, DOI : 10.1145/358669.358677.

Trevennor, Alan (2012), « A Brief History of Microcontrollers », in : *Practical AVR Microcontrollers : Games, Gadgets, and Home Automation with the Microcontroller Used in Arduino*, sous la dir. d'Alan Trevennor, Berkeley, CA : Apress, p. 3-11, DOI : 10.1007/978-1-4302-4447-9_1.

Triplett, Jack E. (1999), « The Solow Productivity Paradox : What Do Computers Do to Productivity ? », in : *The Canadian Journal of Economics / Revue canadienne d'Economique* 32.2, p. 309-334, DOI : 10.2307/136425.

Troelstra, Anne S. (1969), *Principles of Intuitionism*, Lecture Notes in Mathematics 95, Berlin : Springer, DOI : 10.1007/BFb0080643.

Turing, Alan M. (1937), « On Computable Numbers, with an Application to the Entscheidungsproblem », in : *Proceedings of the London Mathematical Society* s2-42.1, p. 230-265, DOI : 10.1112/plms/s2-42.1.230.

— (1950), « Computing Machinery and Intelligence », in : *Mind* 59.236, p. 433-460.

Turner, Raymond (2010), « Programming Languages as Mathematical Theories », in : *Thinking Machines and the Philosophy of Computer Science*, sous la dir. de Jordi Vallverdú, Hershey : Information Science Reference, p. 66-82.

— (mai 2011), « Specification », in : *Minds and Machines* 21.2, p. 135-152, DOI : 10.1007/s11023-011-9239-x.

— (2018), *Computational Artifacts : Towards a Philosophy of Computer Science*, Berlin : Springer.

Tymoczko, Thomas, éd. [1986] (1998), *New Directions in the Philosophy of Mathematics : An Anthology*, revised and expanded paperback ed., Princeton : Princeton University Press.

Uckelman, Sara L. (2010), « Computing with Concepts, Computing with Numbers : Llull, Leibniz, and Boole », in : *Conference on Computability in Europe*, Berlin : Springer, p. 427-437.

Univalent Foundations Program (2013), *Homotopy Type Theory : Univalent Foundations of Mathematics*, Princeton, URL : http://homotopytypetheory.org/book/.

Usher, Abbott Payson et Jean-Yves Dupont (1988), *A History of Mechanical Inventions*, Revised edition, New York : Dover.

Utkucu, Duygu et Hatice Sözer (août 2020), « Interoperability and Data Exchange within BIM Platform to Evaluate Building Energy Performance and Indoor Comfort », in : *Automation in Construction* 116, p. 103225, DOI : 10.1016/j.autcon.2020.103225.

Uzan, Pierre (nov. 2007), « Physique, information statistique et complexité algorithmique », in : *Philosophia Scientiæ* 11-2, p. 121-162, DOI : 10.4000/philosophiascientiae.347.

Vaccari, Andrés (2008), « Legitimating the Machine : The Epistemological Foundation of Technological Metaphor in the Natural Philosophy of René Descartes », in : *Philosophies of Technology : Francis Bacon and His Contemporaries*, sous la dir. de Claus Zittel et al., Intersections vol. 11, Leiden : Brill, p. 287-336.

Van Kerkhove, Bart (2009), *New Perspectives on Mathematical Practices : Essays in Philosophy and History of Mathematics*, World Scientific.

Van Leeuwen, Jan, éd. (1990), *Handbook of Theoretical Computer Science*, Amsterdam : Elsevier.

Vandendriessche, Éric (2016), « Des Pratiques Algorithmiques et Géométriques Propres à Des Sociétés Autochtones », in : *Actes Du Colloque Du Groupe de Didactique Des Mathématiques Du Québec*, Ottawa, p. 11-26.

Vandendriessche, Éric et Céline Petit (nov. 2017), « Des prémices d'une anthropologie des pratiques mathématiques à la constitution d'un nouveau champ disciplinaire : l'ethnomathématique », in : *Revue d'histoire des sciences humaines* 31, p. 189-219, DOI : 10.4000/rhsh.458.

Vander Haeghen, Georges [1957] (2015), *760 mouvements mécaniques : classés pour favoriser les inventions*, Sciences & techniques anciennes, Saint-Laurent-le-Minier : Decoopman.

Varadi, Mihaly et al. (jan. 2022), « AlphaFold Protein Structure Database : Massively Expanding the Structural Coverage of Protein-Sequence Space with High-Accuracy Models », in : *Nucleic Acids Research* 50.D1, p. D439-D444, DOI : 10.1093/nar/gkab1061.

Varela, Francisco J., Evan Thompson et Eleanor Rosch [1991] (2016), *The Embodied Mind : Cognitive Science and Human Experience*, Revised edition, Cambridge, Mass : MIT Press.

Varenne, Franck (avr. 2009), « Simulation informatique et pluriformalisation des objets composites », in : *Philosophia Scientiæ* 13-1, p. 135-154, doi : 10.4000/philosophiascientiae.79.

— (2012), *Théorie, réalité, modèle. Epistémologie des théories et des modèles face au réalisme dans les sciences*, Sciences & philosophie, Paris : Editions Matériologiques.

— (2013), « Modèles et Simulations : Pluriformaliser, Simuler, Rematématiser », in : *Modéliser & Simuler. Epistémologies et Pratiques de La Modélisation et de La Simulation, Tome 1*, sous la dir. de Franck Varenne et Marc Silberstein, p. 297-325.

— (2019), « About the Persistence and Varieties of 'Materiality Arguments' around the Machine : From the Programs Verification's Debate to the Debate about the Computer Simulations' Empiricity », in : *PROGRAMme Spring Workshop*, Lille.

Vera, Alonso H. et Herbert A. Simon (jan. 1993a), « Situated Action : Reply to Reviewers », in : *Cognitive Science* 17.1, p. 77-86, doi : 10.1207/s15516709cog1701_6.

— (1993b), « Situated Action : Reply to William Clancey », in : *Cognitive Science* 17.1, p. 117-133, doi : 10.1207/s15516709cog1701_8.

— (avr. 1994), « Reply to Touretzky and Pomerleau : Reconstructing Physical Symbol Systems », in : *Cognitive Science* 18.2, p. 355-360, doi : 10.1207/s15516709cog1802_6.

Vérin, Hélène (1993), *La gloire des ingénieurs : l'intelligence technique du XVIe au XVIIIe siècle*, L'évolution de l'humanité, Paris : Albin Michel.

— (2008), « Rédiger et réduire en art : un projet de rationalisation des pratiques », in : *Réduire en art : la technologie de la Renaissance aux Lumières*, sous la dir. de Hélène Vérin et Pascal Dubourg Glatigny, Paris : Éditions de la Maison des sciences de l'homme.

Vérin, Hélène et Pascal Dubourg Glatigny, éd. (2008), *Réduire en art : la technologie de la Renaissance aux Lumières*, Paris : Éditions de la Maison des sciences de l'homme.

Vermaas, Pieter E. et Wybo Houkes (2006), « Technical Functions : A Drawbridge between the Intentional and Structural Natures of Technical Artefacts », in : *Studies in History and Philosophy of Science Part A*, The Dual Nature of Technical Artefacts 37.1, p. 5-18, doi : 10.1016/j.shpsa.2005.12.002.

Vincenti, Walter G. (1992), « Engineering Knowledge, Type of Design, and Level of Hierarchy : Further Thoughts about What Engineers Know… », in : *Technological Development and Science in the Industrial Age*, Berlin : Springer, p. 17-34.

Voevodsky, Vladimir (juin 2015), « An Experimental Library of Formalized Mathematics Based on the Univalent Foundations », in : *Mathematical Structures in Computer Science* 25.5, p. 1278-1294, doi : 10.1017/S0960129514000577.

von Neumann, John [1945] (1993), « First Draft of a Report on the EDVAC », in : *IEEE Annals of the History of Computing* 15.4, p. 27-75, doi : 10.1109/85.238389.

Von Plato, Jan (1995), « The Axioms of Constructive Geometry », in : *Annals of pure and applied logic* 76.2, p. 169-200.

Vorms, Marion (2009), « Théories, modes d'emploi : une perspective cognitive sur l'activité théorique dans les sciences empiriques », thèse de doctorat, Paris : Université Paris 1, url : https://tel.archives-ouvertes.fr/tel-00462403/document (visité le 19/10/2017).

Wadler, Philip (nov. 2015), « Propositions as Types », in : *Communications of the ACM* 58.12, p. 75-84, doi : 10.1145/2699407.

Wagner, Pierre (1998), *La machine en logique*, Paris : Presses universitaires de France.

Walras, Léon [1874] (1988), *Éléments d'économie politique pure : ou théorie de la richesse sociale*, éd. établie et trad. par Claude Mouchot, Oeuvres économiques complètes / Auguste et Léon Walras 8, Paris : Economica.

Walsh, Kirsten (2012), « Did Newton Feign the Corpuscular Hypothesis ? », in : *Rationis Defensor : Essays in Honour of Colin Cheyne*, sous la dir. de James Maclaurin, Studies in History and Philosophy of Science, Dordrecht : Springer, p. 97-110, doi : 10.1007/978-94-007-3983-3_8.

Wang, Hao [1961] (1998), « Theory and Practice in Mathematics », in : *New Directions in the Philosophy of Mathematics : An Anthology*, sous la dir. de

Thomas Tymoczko, revised and expanded paperback ed., Princeton : Princeton University Press, p. 129-152.

Wang, Xiaowei (2016), « Towards an Ontology of Software », thèse de doctorat, Trento : Università di Trento.

Wang, Yanjing (2018), « A Logic of Goal-Directed Knowing How », in : *Synthese* 195.10, p. 4419-4439, doi : 10.1007/s11229-016-1272-0.

Warin, Isabelle (fév. 2022), « La notion de technè en Grèce ancienne », in : *Artefact. Techniques, histoire et sciences humaines* 15, p. 43-60, doi : 10.4000/artefact.11251.

Warren, Michael (sept. 2021), *How Many Developers Are There in the World 2022 ?*, url : https://www.bairesdev.com/blog/how-many-software-developers-in-the-world/ (visité le 25/01/2023).

Wegner, Peter (mai 1997), « Why Interaction Is More Powerful than Algorithms », in : *Communications of the ACM* 40.5, p. 80-91, doi : 10.1145/253769.253801.

Weizenbaum, Joseph (1976), *Computer Power and Human Reason : From Judgment to Calculation*, San Francisco : W. H. Freeman and Company.

Welsh, Matt (jan. 2023), « The End of Programming », in : *Communications of the ACM* 66.1, p. 34-35, doi : 10.1145/3570220.

West, A. F. et H. D. Thompson (1891), « On Dulcarnon, Elefuga, and Pons Asinorum as Fanciful Names for Geometrical Propositions », in : *Princeton College Bulletin* 3, p. 84-88.

Westfall, Richard S. et Itzhak Torchin (1977), *The Construction of Modern Science : Mechanisms and Mechanics*, History of Science, Cambridge : Cambridge University press.

Wetter, Michael et al. (juin 2011), « Recent Developments of the Modelica "Buildings" Library for Building Energy and Control Systems », in : doi : 10.3384/ecp11063266.

Wieringa, Roel (mai 2009), « Design Science as Nested Problem Solving », in : *Proceedings of the 4th International Conference on Design Science Research in Information Systems and Technology*, DESRIST '09, New York : ACM Books, p. 1-12, doi : 10.1145/1555619.1555630.

Wiggins, David (1975), « Deliberation and Practical Reason », in : *Proceedings of the Aristotelian Society*, t. 76, p. 29-51.

Wigner, Eugene P. (1960), « The Unreasonable Effectiveness of Mathematics in the Natural Sciences », in : *Communications on Pure and Applied Mathematics* 13.1, p. 1-14.

Wilkes, Maurice (1979), *The Birth and Growth of the Digital Computer*, url : https://www.youtube.com/watch?v=MZGZfsr1KfY (visité le 16/08/2022).

Wing, Jeannette M. (2006), « Computational Thinking », in : *Communications of the ACM* 49.3, p. 33-35.

Winsberg, Eric (2015), « Computer Simulations in Science », in : *The Stanford Encyclopedia of Philosophy*, sous la dir. d'Edward N. Zalta, url : https://plato.stanford.edu/archives/sum2015/entries/simulations-science/ (visité le 23/10/2017).

Wittgenstein, Ludwig [1953] (jan. 2014), *Recherches philosophiques*, trad. par Françoise Dastur et al., Tel, Paris : Gallimard.

Wolff, Christian [1729] (2019), *Über den Unterschied zwischen dem systematischen und dem nicht-systematischen Verstand : Lateinisch - Deutsch*, sous la dir. de Michael Albrecht, Philosophische Bibliothek Band 710, Hamburg : Felix Meiner Verlag.

Wu, Lifang et al. (août 2016), « Smart Supply Chain Management : A Review and Implications for Future Research », in : *The International Journal of Logistics Management* 27.2, p. 395-417, doi : 10.1108/IJLM-02-2014-0035.

Younes, Nadja et Ulf-Dietrich Reips (mars 2019), « Guideline for Improving the Reliability of Google Ngram Studies : Evidence from Religious Terms », in : *PLOS ONE* 14.3, e0213554, doi : 10.1371/journal.pone.0213554.

Yudiansyah, Adrian et al. (déc. 2020), « Can The Mobile Robot Be a Future Order-Picking Solution ?: A Case Study at Amazon Fulfillment Center », in : *Advances in Transportation and Logistics Research* 3.0, p. 800-806, doi : 10.25292/atlr.v3i0.339.

Zeuthen, Hieronymus Georg (1896), « Die Geometrische Construction Als "Existenzbeweis" in Der Antiken Geometrie », in : *Mathematische Annalen* 47.2, p. 222-228.

Zwart, Sjoerd D. (déc. 2020), « Prescriptive Engineering Knowledge », in : *The Routledge Handbook of the Philosophy of Engineering*, sous la dir. de Diane P. Michelfelder et Neelke Doorn, New York : Routledge, p. 111-126, doi : 10.4324/9781315276502.

Zwart, Sjoerd D., Maarten Franssen et Peter A. Kroes (2018), « Practical Inference – A Formal Analysis », in : *The Future of Engineering*, Berlin : Springer, p. 33-52.

Zweig, Janet (sept. 1997), « Ars Combinatoria », in : *Art Journal* 56.3, p. 20-29, doi : 10.1080/00043249.1997.10791829.

Table des matières