HAL
open science

Modélisation de la notion de test de programmes; application à la production de jeux de tests

Luc Bougé

To cite this version:

Luc Bougé. Modélisation de la notion de test de programmes; application à la production de jeux de tests. Génie logiciel [cs.SE]. Université Pierre et Marie Curie - Paris VI, 1982. Français. NNT : . tel-00416558

HAL Id: tel-00416558

https://theses.hal.science/tel-00416558v1

Submitted on 14 Sep 2009

HAL is a multi-disciplinary open access archive for the deposit and dissemination of scientific research documents, whether they are published or not. The documents may come from teaching and research institutions in France or abroad, or from public or private research centers.

L'archive ouverte pluridisciplinaire **HAL**, est destinée au dépôt et à la diffusion de documents scientifiques de niveau recherche, publiés ou non, émanant des établissements d'enseignement et de recherche français ou étrangers, des laboratoires publics ou privés.

T H E S E présentée

pour l'obtention

du

DIPLOME de DOCTEUR de 3e CYCLE

à

L'UNIVERSITÉ PIERRE ET MARIE CURIE
- Paris 6 -

spécialité : Mathématiques

mention : Informatique

par Mr Luc BOUGE

Sujet de la thèse :

MODELISATION DE LA NOTION DE TEST DE PROGRAMMES
APPLICATION A LA PRODUCTION DE JEUX DE TESTS

soutenue le 8 Octobre 1982 devant la Commission composée de :

M B. ROBINET	Président
Mme M.C. GAUDEL	examinateur
M M. NIVAT	«
M J.P. JOUANNAUD	«
M H. GALLAIRE	«
M	«
M J.C. RAULT	invité
M. K. CULIK II	"

à dominique,
douce

à Jérémie,
sourires

RESUME

Ce travail présente une modélisation originale de la notion de test de programmes, à partir de la logique égalitaire du premier ordre. Plusieurs applications concernant notamment les types abstraits algébriques et la validation automatique de spécifications sont proposées.

A partir d'une étude intuitive de la notion de test, nous dégageons la notion de processus de test, fondée sur le principe de couplage.

Nous définissons la notion de jeu de tests sur un contexte de test. Ses propriétés mathématiques sont étudiées : fiabilité, validité, absence de biais, acceptabilité. Plusieurs ordres partiels sont définis : finesses absolue et asymptotique. Les équivalences déduites donnent lieu à des théorèmes importants.

Une méthode pratique de test est construite à partir de cette théorie, et appliquée à un programme de tri. Cette méthode est particulièrement adaptée à la validation d'un axiome d'un type abstrait algébrique sur une algèbre. Un exemple est présenté, et l'implantation d'un outil automatique de validation de spécification utilisant cette méthode, réalisée à titre expérimental, est décrite.

De nombreuses annexes sont jointes : un résumé des travaux antérieurs sur le problème, une bibliographie sur la validation de programmes par test, une introduction à la logique du premier ordre et un listage partiel de l'implantation réalisée.

TABLE DES MATIERES

Chapitre 0. Introduction

Chapitre 1. Notion de test : approche intuitive

1.1. Test et mathématiques
1.2. Test et vie quotidienne
1.3. Test et logiciel
1.4. Test et bases de données
1.5. Test et statistiques
1.6. Conclusion

Chapitre 2. Modélisation

2.1. Notion de processus de test
2.2. Choix d'un outil mathématique
2.3. Contexte et jeu
2.4. Conclusion

Chapitre 3. Propriétés fondamentales

3.1. Notion de fiabilité
3.2. Notion de validité
3.3. Notion de biais
3.4. Notion d'acceptabilité
3.5. Conclusion

Chapitre 4. Hiérarchies et équivalences

4.1. Finesse absolue
4.2. Finesse asymptotique
4.3. Puissance
4.4. Equivalences
4.5. Conclusion

Chapitre 5. Construction de jeux de tests : outils théoriques et stratégie.

5.1. Restriction de contexte
5.2. Reduction de problème
5.3. Optimisation
5.4. Conclusion

Chapitre 6. Construction de jeux de tests : un exemple

6.1. Phase de représentation
6.2. Phase de construction
6.3. Phase d'application
6.4. Conclusion

Chapitre 7. Test et types abstraits algébriques

7.1. Position du problème et représentation
7.2. Phase de construction
7.3. Phase d'application et optimisation
7.4. Conclusion

Chapitre 8. Validation de spécifications abstraites par test

8.1. Principes fondamentaux
8.2. Description du jeu de tests
8.3. Notion de candidat
8.4. Détails d'implantation
8.5. Conclusion

Chapitre 9. Conclusion

9.1. De la modélisation
9.2. Du développement mathématique
9.3. Des applications pratiques
9.4. De la notion de qualité
9.5. Envoi

ANNEXES

Annexe 1 : Historique de la théorie du test

Annexe 2 : Bibliographie sur la validation de programmes par test

Annexe 3 : Introduction à la logique du premier ordre

Annexe 4 : Listages

Introduction

La notion de test remonte, semble-t-il, aux âges les plus reculés de l'humanité. C'est en effet à cette époque que certaines tribus découvrirent, par hasard probablement, qu'il est possible d'allumer un feu par le frottement rapide et prolongé de deux pièces de bois sec l'une contre l'autre.

Mais ce n'était alors qu'une découverte empirique non communicable.

Un jour se présenta un homme. Il les vit faire, et essaya à son tour en se disant par devers lui :

" Si cette expérience réussit, alors il sera possible à ma tribu d'obtenir du feu par ce moyen, à chaque fois qu'elle le souhaitera ".

L'expérience réussit.

La guerre du feu venait de prendre fin, et la notion de test était née.

La notion de test, comme le soulignaient en 1975 J.B. GOODENOUGH & S.L. GERHART [G&G 75] est sans doute, des notions essentielles en informatique, la moins étudiée.

Elle est essentielle car, jusqu'à présent, le test est le seul moyen <u>efficace</u> d'assurer la correction d'un programme de complexité réaliste (cf. [Boug 82]), plus précisément de <u>valider</u> ces programmes. Il le restera probablement, au moins aussi longtemps que les méthodes formelles de spécification et de preuve ne seront raisonnablement utilisables que pour des programmes de quelques dizaines, voire de quelques centaines de lignes.

Il est raisonnable d'affirmer que les théoriciens de l'informatique, après avoir fourni un immense travail de formalisation autour de la notion de preuve et de celle de spécification, commencent juste à s'apercevoir de l'importance pratique de celle de test.

La notion de test en est aujourd'hui au stade où se trouvait, il y a vingt années, celle de langage de programmation, alors que la différence entre syntaxe et sémantique commençait à peine à être perçue.

La notion de langage de programmation n'a pu atteindre la précision qu'elle possède aujourd'hui que par la convergence d'un grand nombre de travaux théoriques, venus d'horizons divers.

Des outils mathématiques nouveaux ont été créés, de nouveaux concepts ont été définis, des méthodes ont été décrites. L'introduction d'un formalisme mathématique rigoureux, fondé sur des notions abstraites bien connues, depuis longtemps reconnues et largement acceptées, a été décisive dans cette évolution.

Nous pensons qu'il peut en être ainsi de la notion de test.

C'est l'objet de cette thèse.

Une première difficulté est l'état confus et imprécis du vocabulaire et de l'intuition quant à la notion de test. Notre premier chapitre a pour but de <u>préciser</u> le sens de certains mots et de <u>dégager</u> les concepts fondamentaux liés à la notion de test.

Une fois ce travail de mise au point réalisé, nous proposons une <u>modélisation</u> de la notion de test dans le cadre de la <u>logique du premier</u>

ordre égalitaire. Cette modélisation permet de préciser les rapports entre la notion de test, le correct, le "probablement correct" et l'incorrect, qui font la spécificité et la complémentarité de cette notion vis à vis de celle de preuve. Nous introduisons les notions de contexte de test et de jeu de tests.

Nous passons ensuite à l'étude théorique, mathématique de ce modèle. Au chapitre troisième, nous dégageons et définissons rigoureusement les propriétés fondamentales d'un jeu de tests sur un contexte que sont la validité et l'absence de biais. Ceci nous conduit à définir ce qu'est, à notre sens, un "bon" jeu de tests (acceptabilité).

Après avoir défini ce qu'est un "bon" jeu de tests, il nous faut préciser ce qu'est un jeu de tests "meilleur" qu'un autre. C'est l'objet du quatrième chapitre qui culmine avec le "Théorème fondamental du test" (cf. 4.4.3) qui décrit la répartition des "bons" jeux de tests sur un contexte donné.

Cette modélisation est non seulement fructueuse et élégante au plan théorique. Elle nous permet aussi de décrire une méthode effective et rigoureuse de construction de jeux de tests. Le chapitre cinquième présente les outils utilisés par cette méthode,

et esquisse une description de celle-ci.

Le chapitre sixième montre l'application de la méthode précédente au test d'un programme de tri. En particulier, notre formalisme permet une approche rigoureuse du problème de l'optimisation du jeu de tests produit.

Pour de nombreuses raisons, le formalisme des types abstraits algébriques offre un terrain d'application privilégié pour notre travail. A titre d'illustration, nous étudions le test d'un axiome du type File sur une File (Entier)-algèbre. C'est l'objet du chapitre septième.

L'exemple proposé se généralise en fait au test d'un axiome sur une algèbre. Le chapitre huitième propose la description d'un outil de validation automatique par test d'une spécification algébrique sur une algèbre. Une version expérimentale de cet outil est actuellement implantée.

Plusieurs annexes sont jointes à ce travail.

La première retrace les principales tentatives de théorisation de la notion de test qui nous ont inspiré.

La seconde propose une bibliographie thématique sur le problème de la validation de

programmes par test. Ces deux annexes sont extraites de [Boug 82].

Une troisième annexe propose une <u>introduction à la logique du premier ordre</u> présentant la plupart des outils utilisés au cours de notre travail.

Une quatrième annexe fournit un <u>listage partiel</u> de l'outil décrit au chapitre huitième.

Chapitre 1 : Notion de test : approche intuitive.

Avant d'aborder la description de la modélisation mathématique que nous proposons pour la notion de test, plus précisément la notion de processus de test, il nous semble important de proposer une approche intuitive, empirique de cette notion.

Cette approche est nécessaire pour plusieurs raisons.

Tout d'abord, le mot "test" est d'un usage trop courant, trop répandu pour avoir une acception bien définie et largement admise. Même dans le domaine restreint de l'informatique, ce mot recouvre de nombreuses réalités, contrairement au mot "preuve", qui, bénéficiant des nombreuses études théoriques dont il fait l'objet, possède maintenant un sens précis. Ainsi, par exemple, le mot "test" désigne aussi bien le <u>processus</u> consistant à tester un programme, que chacune des <u>expériences</u> élémentaires accomplies ou proposées au cours de ce processus.

D'autre part, il nous semble fondamental que le travail que nous entreprenons ne soit

pas simplement l'une de ces constructions théoriques entièrement coupées de toute réalité concrète dont de nombreux mathématiciens se délectent quotidiennement. Notre travail est un travail de modélisation d'une notion qui, bien que jusqu'à présent largement empirique, a déjà ses lettres de noblesses. Ce chapitre a pour but de dégager, d'esquisser un certain "esprit" de la notion de test, auquel nous essaierons de rester fidèle tout au long des développements théoriques.

Enfin, il nous paraît important, pour aborder ce travail de modélisation, de mettre en relation la notion de test avec d'autres notions, celle de preuve formelle notamment, mais aussi de qualité, de rentabilité, d'hypothèse implicite. Toute d'une telle mise en perspective, la notion de test a trop souvent été conçue comme un cas particulier d'autres notions, celle de preuve de correction entre autre. Nous tensons, au contraire, par ce travail, dégager une spécificité de la notion de test par rapport aux notions connexes.

Dans cette optique, nous allons mettre le mot "test" en situation ; nous étudions l'utilisation de ce mot en mathématiques,

dans certains magazines spécialisés, dans les domaines du logiciel et du matériel (cf. [Boug 82]), dans le domaine des bases de données, et enfin en théorie des probabilités et statistiques.

1.1 Test et mathématiques.

Dans cette partie, nous étudions l'emploi du mot "test" en mathématiques, et plus particulièrement en algèbre (théorie des langages). La notion de test à laquelle il y est fait référence alors est en général très pauvre, puisqu'il ne s'agit que d'une abréviation signifiant "ensemble fini de valeurs du domaine".

Cette notion de test se rapproche en fait beaucoup de celle proposée p GOODENOUGH & GERHART en [GG 75] (cf. annexe 1) dans le cadre de la validation de programmes par test. Un test T d'un programme P opérant sur un domaine D est alors simplement une partie (finie) de D. Ce test devrait être tel que si P est correct sur T, alors P l'est sur D.

Considérons, par exemple, l'emploi de la notion de test dans [Culi 80]. Nous y trouvons la définition suivante :

<u>définition</u> :

Nous dirons qu'un sous-ensemble fini F d'un langage L est un test pour L si, pour tout couple d'homomorphismes (g,h)

$g(x) = h(x)$ pour tout x de L
si et seulement si
$g(x) = h(x)$ pour tout x de F

Il est alors possible d'énoncer le théorème suivant.

<u>théorème</u> 6.3
Pour tout langage acontextuel, il existe effectivement un test.

1.1.1. test et critère de sélection

Nous retrouvons, dans la définition précédente, le concept extrêmement pertinent de <u>critère de sélection</u> introduit par GOODENOUGH & GERHART notamment, et qui sous-tend toute la théorie qu'ils proposent (cf. annexe 1).

En effet, n'importe quel sous-ensemble (fini) de L ne constitue pas un test. Mais l'important n'est pas tant d'exhiber un test F que de définir la propriété devant être vérifiée par un sous-ensemble (fini) de L pour constituer un test.

Avec les notations de [G&G 75], la définition décrit un prédicat, un critère de selection sur les sous-ensembles finis de L :

COMPLET (F) =

$\forall f \forall g \left([\forall x \in F \ g(x) = h(x)] \leftrightarrow [\forall x \in L \ g(x) = h(x)] \right)$

Si COMPLET(F) est vrai, alors F est un test.

Remarquons la forme très particulière de ce prédicat. Nous appellerons par la suite (cf. 2.4.2.1, 2.4.2.2, 3.4.3.2) une telle propriété une <u>hypothèse d'uniformité</u> : relativement à $g(x) = h(x)$, L est uniforme par rapport à F

1.1.2. Test et preuve

L'emploi de la notion de test dans [Culi 80] permet de mieux cerner les rapports entre test et preuve. Il est bien évident que, dans ce cas, trouver un test F et vérifier, pour chaque x de F, $g(x) = h(x)$, constitue une preuve parfaitement rigoureuse de la propriété $\forall x \in L, \ g(x) = h(x)$

Cependant, le fait d'introduire la notion de test revient à assurer une propriété plus forte que cette simple formule. Cette propriété concerne non pas les objets en eux mêmes, mais les règles formelles permettant de manipuler ces objets. C'est donc une <u>méta-</u>

propriété sur le prédicat exprimant que telle formule est démontrable par telle méthode (le prédicat "demo" de KOWALSKI)

En effet, le théorème montre que la preuve de la propriété $\forall x \; g(x) = h(x)$ peut se faire, de manière effective, selon un schéma très particulier.

Dans un premier temps, trouver des éléments (des individus) $x_1, \ldots, x_n$ de L, et <u>démontrer</u> le théorème

$$\forall g \; \forall h \; [\, g(x_1) = h(x_1) \wedge \ldots \wedge g(x_n) = h(x_n) \rightarrow \forall x \; g(x) = h(x) \,]$$

Dans un deuxième temps, <u>vérifier</u> chacune des formules :

$$g(x_i) = h(x_i)$$

pour $1 \leq i \leq n$.

Ce que le théorème assure, c'est que la preuve, de manière effective, se factorise en deux parties.

Une partie est d'ordre purement <u>formel</u> et <u>axiomatique</u>, ne faisant intervenir que les axiomes vérifiés par L et non L lui-même. Cette partie est <u>inductive</u>, <u>transcendante</u> en ce sens que la formule relie le comportement d'une infinité de valeurs à celui d'un nombre fini de valeurs.

La seconde partie est d'ordre expérimental, se réduisant en fait à un calcul sur des valeurs

1.1.3. Conclusion

De cette partie concernant le test en mathématique, nous retiendrons l'importance de la notion de critère de sélection, puisque celle-ci est présente même dans une acception pauvre du mot "test".

Nous retiendrons aussi que l'introduction de la notion de test lors de la preuve d'une propriété revient à poser une "méta-assertion" sur celle-ci, puisqu'elle conduit à la factoriser en une partie transcendante et une partie expérimentale. La première démontre axiomatiquement que le résultat de certaines expériences permet d'induire la propriété sous test. La seconde réalise effectivement ces expériences.

1.2. Test et vie quotidienne

" Test : choisissez votre magnétophone " ;
" Test : les machines à laver " ;
" Un test exclusif : saurez-vous planter les choux ! " ;
" Les secrets des tests psychotechniques " ;
" Les tests : Que sais-je n° 76 " ;

voici quelques titres glanés au hasard des couvertures que chacun peut rencontrer chaque jour au coin de sa rue, chez son libraire habituel.

Il est bien certain que le mot " test " y recouvre des réalités très diverses, parfois assez éloignées de l'acception informatique de ce mot. Cependant, il nous semble qu'un examen attentif de ces réalités peut nous aider à mieux cerner notre problème.

Dans tous les cas, le mot " test " est lié, plus ou moins explicitement, à une <u>propriété</u> et à un <u>domaine</u>. Par exemple, dans le premier cas,

" Test : choisissez votre magnétophone ",

la propriété est le fait, pour un magnétophone, d'être le mieux adapté à vos besoins,

le domaine est celui de tous les magnétophones possibles.

Autrement dit, si S est l'ensemble de tous les magnétophones, nous considérons le prédicat est_le_meilleur (x) sur S. Le problème est de déterminer si, étant donné un magnétophone x_0, est_le_meilleur (x_0).

Dans le second cas, l'on s'intéresse sans doute à la perfection technique des machines à laver de marque X. La propriété considérée est que les machines à laver X sont de bonnes machines à laver.

L'on travaille donc sur le domaine S des machines à laver de marque X, avec un prédicat est_bon sur S. La propriété sous test est de la forme : $\forall x$ est_bon (x).

Pourquoi une propriété a-t-elle besoin d'être testée ? En général, c'est parce qu'elle n'est pas directement expérimentable, <u>effectivement vérifiable</u> par l'intermédiaire d'une ou d'un nombre fini d'expériences. Une expérience n'est rien d'autre qu'une sous-propriété <u>effectivement vérifiable</u>. Faire une expérience, c'est vérifier que la propriété la constituant est vraie. Par définition, il existe un algorithme permettant de répondre

en un temps fini à cette question.

Considérons le premier cas. Pour savoir si x_0 est le meilleur magnétophone pour vous, il vous faudrait essayer tous les magnétophones disponibles durant un temps potentiellement infini. La propriété est_le_meilleur (x_0) n'est donc pas effectivement vérifiable, et nécessite d'être testée.

Considérons le second cas. Un laboratoire indépendant pourra, par l'examen approfondi d'une machine à laver x_0, vous dire si est_bon(x_0). Mais il ne peut examiner toutes les machines à laver de marque x.

Nous appellerons un problème analogue à celui du premier cas un problème <u>temporel</u>. Un problème analogue à celui du second cas sera dit <u>spatial</u>

1.2.1 Propriétés temporelles

Considérons un domaine S, et un prédicat P sur S. P est dit <u>temporel</u>, s'il n'est pas décidable, en un temps fini, de conclure si $P(x_0)$ ou non, x_0 étant fixé dans S. P n'est pas effectivement vérifiable sur S.

Pour tester la propriété $P(x_0)$, nous serons donc amené à passer par l'intermédiaire d'un autre prédicat Q, tel que, pour tout x de S, $Q(x)$ implique $P(x)$ par exemple (il suffirait que $Q(x_0)$ implique $P(x_0)$).

Autrement dit, Q est tel que

$$\forall x \, [Q(x) \rightarrow P(x)] \qquad (1)$$

est une formule vraie sur le domaine S.

Q est en général plus 'simple' que P, et, le plus souvent, effectivement vérifiable sur S. $Q(x_0)$ est alors une <u>expérience</u> dont la réussite implique $P(x_0)$. Nous avons donc bien testé $P(x_0)$.

Dans notre exemple, $Q(x)$ exprimera peut-être que x a été chaudement recommandé par le vendeur, ou que x n'est pas trop cher, ou qu'il a des petites lampes rouges, jaunes et vertes qui s'allument et s'éteignent avec la musique etc...

Remarquons qu'il se peut très bien que, bien que x_0 eût été pour vous le meilleur choix, $Q(x_0)$ ne soit pas vrai (car "faux implique vrai" est vrai !). Par contre, si $Q(x_0)$ est vrai, alors, par (1), $P(x_0)$ l'est aussi.

1.2.2. Propriétés spatiales (universelles)

Considérons un domaine S, et un prédicat Q sur S. Supposons que Q soit <u>effectivement vérifiable</u> sur S. Pour x dans S, nous pouvons conclure, en un temps fini, si $Q(x)$ ou non. La propriété $\forall x\, Q(x)$ sera dite alors <u>spatiale</u>. Si S est infini, elle n'est pas effectivement vérifiable, mais cependant, elle l'est localement (sur un sous-domaine fini de S).

Pour tester la propriété $\forall x\, Q(x)$, nous devrons <u>induire</u>, à partir de la connaissance locale du comportement de Q, un résultat concernant le comportement global de Q.

Ceci n'est possible que si nous savons par ailleurs que le comportement de Q n'est pas totalement erratique, mais possède une certaine uniformité, une certaine régularité.

Nous appellerons <u>hypothèse de régularité</u> une hypothèse de la forme :

$$Q(x_0) \wedge \dots \wedge Q(x_k) \rightarrow \forall x\, Q(x), \qquad (2)$$

où $x_0, \dots, x_k$ sont des éléments fixés de S.

Nous appellerons <u>hypothèse d'uniformité</u> une hypothèse de la forme :

$$[\exists x \; Q(x)] \rightarrow [\forall x \; Q(x)] \qquad (3)$$

Chacune de ces hypothèses nous fournit un moyen de tester $\forall x \; Q(x)$. Pour (2), nous réaliserons la suite (finie) d'expériences $Q(x_0), \ldots, Q(x_k)$. Pour (3), nous considérerons un élément donné de S, x_0, et réaliserons l'expérience $Q(x_0)$.

Dans notre exemple, les machines à laver X étant produites en série selon les mêmes plans et par les mêmes méthodes, l'hypothèse (3) est vraisemblable. Il suffira donc au laboratoire de tester <u>une seule machine</u> pour pouvoir accorder un label de qualité à toutes les machines de la marque.

1.2.3. Anisotropie et biais

Un point doit retenir notre attention. Dans tous les exemples que nous avons décrits dans cette partie, nous avons, plus ou moins consciemment, privilégié la propriété sous test par rapport à sa négation.

De manière plus précise, l'on s'intéresse à la validité d'une propriété sur un domaine plus qu'à sa non-validité éventuelle.

De manière très générale, cette remarque nous montre que la notion de test est essentiellement <u>anisotrope</u> : la propriété à tester est considérée comme plus importante que sa négation.

Ce caractère anisotrope peut conduire à réaliser des expériences biaisées, comme le montre la remarque terminant le paragraphe 1.2.1.

Dans le cas de cet exemple, en effet, il est, a priori, possible que $P(x_0)$ soit vrai, sans que $Q(x_0)$ le soit. L'expérience $Q(x_0)$ n'est porteuse d'information que si elle réussit, sinon elle ne nous apprend rien sur $P(x_0)$. Si, de l'échec de $Q(x_0)$, nous concluons que $P(x_0)$ est faux, nous commettons une erreur peut-être. Ce test est dit <u>biaisé</u> en ce sens qu'il peut nous faire rejetter la propriété sous test alors qu'elle est vérifiée en fait.

Par contre, au paragraphe 1.2.2., les propriétés (2) et (3) sont en fait des équivalences, et les tests qui s'en déduisent sont <u>non-biaisés</u> : Si l'une des expériences qu'ils nous conduisent à réaliser échoue, nous pouvons affirmer en toute rigueur que $\forall x Q(x)$

est faux.

1.2.4 Conclusion

De cette partie concernant le test dans la vie quotidienne, nous retiendrons la distinction entre propriétés temporelles et spatiales (universelles).

Nous retiendrons aussi le caractère profondément anisotrope de la notion de test qui conduit à privilégier la propriété sous test par rapport à sa négation. Ce caractère peut conduire à considérer des tests biaisés qui peuvent éventuellement déclarer la propriété fausse alors qu'elle est vraie. De tels tests ne doivent pas être considérés comme de "bons" tests. Dans le cas des propriétés spatiales universelles, nous avons montré que les hypothèses de régularité et d'uniformité conduisent à des tests non-biaisés.

1.3. Test et logiciel

Dans cette partie, nous décrivons rapidement la notion de test dans le domaine du logiciel, et plus précisément de la validation de programmes par test. Nous ne faisons que résumer l'étude beaucoup plus large que nous avons par ailleurs consacrée à ce problème ([Boug 82]).
Une bibliographie sur ce problème est proposée en annexe 2 à cette thèse.

Dans le cas de la validation de programmes par test, nous sommes en présence, conformément au vocabulaire proposé dans la partie précédente, d'une propriété spatiale purement universelle ; en effet, le problème est de montrer que le programme (le système) se comporte correctement pour chacune des valeurs de son domaine d'entrée (cf. annexe 1). Ceci peut s'exprimer par la formule :

$$\forall x \in D \quad OK(x),$$

où D est le domaine d'entrée,
et OK le prédicat de correction, exprimant que la donnée x est traitée correctement par le système.

Un test peut alors être vu comme <u>une suite d'expériences</u> consistant à analyser le comportement du programme sur une partie finie de son domaine d'entrée. (cf. annexe 1).

1.3.1. Oracles de correction

Comme le montre [Weyu 80], le prédicat OK est en général difficile à déterminer et à manipuler dans la pratique. Comment décider si un programme censé calculer la millième décimale de π est correct ou non ? Il se peut aussi que la correction du programme dépende d'une telle masse d'informations que, bien que théoriquement trivialement décidable, elle ne le soit pas humainement.

L'on est donc, la plupart du temps, conduit à considérer la propriété OK(x) comme une propriété <u>temporelle</u>, bien qu'elle ne le soit pas en toute rigueur. L'on détermine alors une condition approchée de correction du programme, donnant ainsi naissance à un nouveau prédicat OK' sur D. Sous l'hypothèse :

$$\forall x \in D, \quad OK'(x) \rightarrow OK(x), \qquad (4)$$

l'on peut se ramener au cas d'un prédicat "humainement vérifiable".

Il est important de remarquer que (4) n'est, en général, pas vérifiée par le système sous test : par exemple, si le programme est censé calculer sinus (x), et si OK' lui impose seulement de calculer une valeur proche de sinus (x). Mais, dans les fait, tout se passe comme si (4) était vérifiée. En effet, le programme étant écrit par un humain suivant une spécification et des méthodes humaines, les erreurs qu'il contient sont essentiellement dues à des oublis ou à des défauts conceptuels, conduisant à des comportements très différents du comportement correct pour quelques valeurs.

1.3.2. Stratégies de test

En remplaçant éventuellement le prédicat de correction OK par un prédicat de correction approché, nous sommes donc confronté au problème du test de la propriété

$$\forall x \quad OK(x)$$

sur le domaine D, OK étant effectivement

(humainement) vérifiable sur D.

Selon la méthode décrite en 1.2.8, il nous faut déterminer un sous-ensemble T de D tel que l'hypothèse de régularité

$$[\forall x \in T \quad OK(x)] \rightarrow [\forall x \in D \quad OK(x)] \qquad (5)$$

soit <u>probablement vraie</u>.

Une <u>stratégie de test</u> est une méthode de construction de ce sous-ensemble T. De nombreuses stratégies (cf. [Boug 82]) ont été décrites dans la littérature. Elles dépendent le plus souvent de la structure syntaxique du programme sous test. L'ensemble T considéré est alors de très grande taille, voire de taille infinie; l'on considère donc une approximation finie de T, T'.

Par exemple les stratégies d'analyse des chemins définissent l'ensemble T ainsi. Pour tout chemin du programme, T doit contenir au moins une valeur du domaine d'entrée suivant ce chemin. Si le programme contient des boucles, il y a en général une infinité de chemin. T' sera tel que, par exemple, toutes les boucles soient parcourues au moins de 0 à k fois.

1.3.3. Mesures de qualité

Il est cependant difficile d'appliquer rigoureusement les stratégies théoriques de test à des programmes non triviaux.

D'autre part, étant donné un ensemble de valeur T, l'hypothèse (5) n'est que probablement vraie.

Il est donc important, une fois un ensemble T déterminé, de pouvoir déterminer la qualité du test ainsi décrit. Cette qualité est fonction de la validité de l'hypothèse (5), mais aussi du caractère plus ou moins crucial de la correction du programme sur certaines de ses données.

De nombreuses mesures de qualité ont été proposées (cf. [Boug 82]) dont les plus intéressantes sont fondées sur la notion de modèle de fautes, issue du test en matériel.

Dans ce cas, l'on détermine la qualité du test non pas par rapport au programme lui-même, mais par rapport à une famille de représentations abstraites du programme censées prendre en compte, par effet de couplage, les erreurs présentes

dans le programme. Par exemple, les mesures de qualité par mutation (cf. [Budd 81]) étudient la capacité du test à détecter des fautes prédéterminées introduites artificiellement dans le programme, une à une. Par couplage, si ces fautes sont suffisamment représentatives, les fautes plus complexes seront elles aussi prises en compte. 1.22.

1.3.4. Conclusion

Nous avons, dans cette partie, rapidement décrit les principales notions liées au problème de la validation de programmes par test.

La notion de test, dans ce cadre, est étroitement liée à celles d'oracle de correction, de stratégie et de mesure de qualité.

La notion d'oracle de correction est liée à celle de vérifiabilité effective que nous avons déjà citée en 1.2.

Celle de stratégie est liée à celle de critère de sélection, que nous avons vue en 1.1. La formule (5) définit en fait un prédicat COMPLET (T). Remarquons que, en général, la stratégie employée est asymptotique, en ce sens qu'il faudrait un temps infini pour construire un T tel que COMPLET (T)

Enfin, la notion de mesure de qualité est de loin la plus difficile à cerner. Les outils développés dans cette thèse ne nous permettront pas d'en donner une définition intéressante et fructueuse.

1.4. Test et bases de données

Les parties précédentes nous ont permis de dégager quelques notions connexes à la notion de test. Cependant, ces domaines étant mal délimités et difficilement formalisables parfois, nous n'avons pu aller plus avant dans notre exploration.

Le domaine des bases de données, plus précisément des bases de données relationnelles ([Nico 79], [Demo 82]) nous permet d'aborder plus rigoureusement le problème

1.4.1 Situation du problème

Une base de donnée relationnelle peut être vue comme la donnée d'un langage L (égalitaire) de la logique du premier ordre, et d'une structure $\mathcal{S}$ (finie en général) pour ce langage (cf. annexe 3). Cette structure représente l'état de la base à un instant donné.

Si, par exemple, la base contient la généalogie d'un famille, l'ensemble des individus de la structure sera l'ensemble des membres de la famille. Le langage L

contiendra probablement les symboles "père", "mère", et "est_enfant_de". La structure $\mathcal{S}$ contiendra donc des interprétations de ces symboles définissant ainsi la généalogie de la famille à un instant donné.

Il est bien évident que toute L-structure ne représente pas un état de la base de données. Un certain nombre d'hypothèses formelles doivent être vérifiées à chaque instant t par l'état $\mathcal{S}_t$ de la base à cet instant. Soit A l'ensemble de ces formules. Nous pouvons considérer plutôt que A la L-théorie ayant A pour ensemble d'axiomes non-logiques (cf. annexe 3). L'assertion précédente s'exprime donc par.

$$\forall t \quad \mathcal{S}_t \models A$$

La théorie A constitue la théorie des <u>exigences de cohérence</u> de la base.

Dans le cas de notre exemple, A contiendra sans doute l'axiome :

$$\forall x \, \forall y \quad (\text{père}(x) = y \vee \text{mère}(x) = y \longrightarrow \text{est\_enfant\_de}(x, y))$$

Le problème qui se pose dans la pratique est d'assurer la cohérence de la base au cours de son évolution.

Supposons qu'à l'instant t, $\mathcal{S}_t$ soit cohérent : $\mathcal{S}_t \models A$. Supposons qu'entre les instants t et $t+1$ soit né un enfant a_t. $\mathcal{S}_{t+1}$ a pour ensemble de base (univers) celui de $\mathcal{S}_t$ réuni à $\{a_t\}$, et les interprétations des symboles sont inchangées sur l'univers de $\mathcal{S}_t$.

Pour <u>certaines</u> formules ϕ, on sait donc, <u>a priori</u>, que si $\mathcal{S}_t \models \forall x\, \phi(x)$, alors

$$\mathcal{S}_{t+1} \models \forall x\, [x \neq a_t \rightarrow \phi(x)].$$

Par exemple, cela est vrai si

$$\phi(x) = \text{est\_enfant\_de}(x, \text{père}(x))$$

Le problème de la vérification de

$$\forall x\, \phi(x)$$

sur $\mathcal{S}_{t+1}$, difficile si la famille en question est très étendue, se factorise donc en fait en

- une partie <u>expérimentale</u>

$$\mathcal{S}_{t+1} \models \phi(a_t)$$

- une partie <u>transcendante</u>

$$\{\phi(a_t),\ \forall x\, [x \neq a_t \rightarrow \phi(x)]\} \vdash \forall x\, \phi(x)$$

(cf. annexe 3 pour les symboles "$\models$" et "$\vdash$")

Nous retrouvons donc bien la notion de test décrite dans la partie 1.1.

A la partie expérimentale est associée le symbole $\models$ (validité <u>logique</u>) ; à la transcendante, le symbole $\vdash$ (preuve <u>formelle</u>).

1.4.2. Jeu de tests

Le problème n'est en général pas aussi simple, et la vérification de la cohérence de la base à l'instant t+1, même sous l'hypothèse de sa cohérence à l'instant t et connaissant la transformation conduisant de $\mathcal{F}_t$ à $\mathcal{F}_{t+1}$ (transformation élémentaire), peut être extrêmement complexe, et donc coûteuse.

D'autre part, l'utilisation de la base peut être telle qu'elle ne nécessite pas une cohérence parfaite de celle-ci, et qu'un certain "taux d'incohérence" est admissible.

Il y a donc un compromis à trouver entre le prix de la vérification de la cohérence et la qualité de cette vérification. Le problème est donc un problème de rentabilité.

De manière très générale, la vérification complète de la cohérence ne sera pas rentable, et l'on choisira un compromis intermédiaire. Cependant, l'univers de la base étant fini en général, cette éventualité est toujours présente ; ce ne serait même pas le cas si la base était infinie.

L'idée sous-jacente à ces remarques est celle proposée par GOODENOUGH & GERHART [G&G 75] sous le nom de <u>critère de sélection</u>. Nous la reprendrons, en la modifiant, sous le nom de <u>jeu de tests</u>.

Ce qui est important, ce n'est pas tant de disposer d'une expérience (ou d'une famille d'expériences finie) permettant de conclure que $\mathcal{S}_{t+1} \models A$ en toute rigueur, par l'intervention éventuelle de "méta-hypothèses". L'importance est de disposer, pour chaque n, d'une expérience (ou d'une famille finie d'expériences) permettant de conclure que $\mathcal{S}_{t+1} \models A$ "à $\frac{1}{n}$ près".

Une telle famille d'expériences (finie) est appelée un <u>test</u>. Ce que nous souhaitons, c'est donc une famille de tests, <u>un jeu de tests</u> $(T_n)_{n \in \mathbb{N}^*}$, tel qu'à chaque T_n soit associé un <u>coût</u> et une <u>qualité</u>. Plus la réussite de T_n apporte d'informations sur $\mathcal{S}_{t+1} \models A$, plus qualité$(n)$ est grand. Si la réussite de T_n assure $\mathcal{S}_{n+1} \models A$, qualité$(n)$ peut être considéré comme infini. La qualité du test consistant à ne rien faire peut être considérée comme nulle.

Disposant d'une certaine fortune, nous choisirons n tel que coût (n) soit inférieur à notre fortune, et tel que qualité (n) soit maximal, ou que le rapport qualité (n) / prix (n) soit maximal. La réussite de T_n nous donnera <u>la meilleure assurance possible</u> dans notre situation de la cohérence de la base à l'instant $t+1$.

Si la base est finie, il existera probablement n_0 tel que qualité $(n_0) = \infty$. Mais, en général, coût $(n_0) \gg$ fortune, et l'existence de n_0 est toute <u>virtuelle</u>.

1.4.3. Fiabilité

Nous sommes maintenant en présence d'un ensemble de tests $(T_n)_{n \in \mathbb{N}}$. Quelles propriétés minimales de cohérence lui impose?

Si nous suivons [G&G 75], nous sommes conduit à considérer le critère $C = \{T_n\}_{n \in \mathbb{N}}$, et à exiger qu'il soit fiable (cf. annexe 1). Il faudrait donc, pour tout n_1 et n_2,

$$\mathcal{S}_{t+1} \models T_{n_1} \text{ si et seulement si } \mathcal{S}_{t+1} \models T_{n_2}.$$

Mais plaçons-nous maintenant dans l'optique de coût, de qualité et de rentabilité définie au paragraphe précédent. Quel intérêt y a-t-il à considérer deux tests dont les résultats sont

équivalents ? Il suffit de conserver le plus rentable des deux pour obtenir un jeu de tests le plus réduit possible, et donc le plus économique.

Donc, la fiabilité au sens précédent n'est pas une propriété désirable pour un jeu de tests, du moins pour l'optique que nous nous sommes fixé. Par contre, l'utilisateur attend vraisemblablement que la qualité (ainsi que le coût, probablement) soit fonction croissante de l'indice du test dans le jeu de tests. Autrement dit, il attend que T_{n+1} soit un test "meilleur" que T_n, et ce, pour tout n.

La notion de "meilleur" signifie que toute l'information recueillie par la <u>réussite</u> de T_n l'aurait été par celle de T_{n+1}. La qualité est en effet liée à la réussite d'un test, et non à son échec.

Plus précisément, la condition de cohérence s'exprimera donc ainsi :

$\forall n$, si $\mathcal{S}_{t+1} \models T_{n+1}$, alors $\mathcal{S}_{t+1} \models T_n$.

La notion classique de fiabilité, proposée en [G&G 75] doit donc être remplacée par une notion de fiabilité hiérarchisée, rendant ainsi <u>partiellement</u> compte de la notion de qualité.

1.4.4 Généralisation

Nous avons donc défini une propriété de fiabilité que devrait vérifier un jeu de tests de la cohérence de la base à l'instant t+1.

Cette propriété fait intervenir explicitement $\mathcal{S}_{t+1}$. Or, dans la pratique, pour des raisons évidentes de rentabilité, l'on ne connaît que partiellement l'état de la base à chaque instant. L'on connaît seulement la suite de transformations élémentaires ayant conduit de l'état initial jusqu'à cet état.

Il est donc hors de question de s'appuyer sur une connaissance fine de $\mathcal{S}_{t+1}$ pour définir ce qu'est un bon test de la cohérence de $\mathcal{S}_{t+1}$. L'on doit pouvoir se contenter d'une connaissance approximative de l'état considéré. Cette connaissance approximative définit une famille $(\mathcal{S})$ de structures <u>éventuelles</u>, parmi lesquelles se trouve celle représentant <u>réellement</u> l'état de la base à l'instant donné; mais nous ne savons pas précisément quelle est cette structure.

Nos définitions doivent donc prendre en compte cette famille, et non la structure inconnue représentant réellement l'état considéré.

La notion de fiabilité hiérarchisée (nous dirons plus tard fiabilité projective, cf. 3.1.) s'exprimera donc ainsi :

pour <u>toute structure de la famille</u> $(\mathcal{S})$
et pour tout n,
si $\mathcal{S} \models T_{n+1}$, alors $\mathcal{S} \models T_n$.

Il est important de remarquer que, dans la pratique, la structure représentant l'état courant ne sera pas membre, en général, de la famille $(\mathcal{S})$ de structures éventuelles. En effet, celles-ci seront déterminées en supposant, par exemple, qu'elles soient issues d'un état cohérent de la base, par une transformation élémentaire (une naissance!). Elles vérifient donc toutes, en reprenant l'exemple de 1.4.1 :

$$\exists y \; \forall x \left[x \neq y \rightarrow \phi(x) \right] \qquad (6)$$

avec $\phi(x) =$ est_enfant_de $(x,$ père $(x))$

Mais, dans la réalité, l'état de la base à l'instant t a été déclaré cohérent selon un processus de test de plus ou moins bonne qualité. Il n'est donc en général que "probablement" cohérent, et, bien que l'état à l'instant $t+1$ en soit issu par une transformation élémentaire, il ne vérifiera (6) que "probablement".

1.4.5. Conclusion

Cette étude très partielle du problème de la cohérence des bases de données (relationnelles) nous a permis de cerner plus précisément la notion de test.

La notion de test est inséparable de celle de jeu de tests. Un jeu de tests est une famille hiérarchisée de tests.

A chaque test de cette famille, nous avons fait correspondre un coût et une qualité. Nous avons montré qu'il est souhaitable que la qualité (et le coût) soit fonction croissante de l'indice du test dans le jeu de tests.

Nous avons donc imposé à un jeu de tests une propriété de "fiabilité hiérarchique" assurant cette exigence, sans pour autant faire référence explicitement à une notion précise de qualité. Cette propriété ne doit pas faire référence à un état déterminé de la base, mais à une famille d'états éventuels.

En général, l'état réel de la base ne se trouve pas dans cette famille. L'important est seulement que tout se passe comme si il s'y trouvait.

1.5 Test et statistiques

Les parties précédentes nous ont permis, de manière plus ou moins empirique, de dégager quelques caractéristiques de la notion de test et de déterminer quelques notions connexes.

Puisque nous nous proposons d'élaborer une modélisation de la notion de test dans le cadre de la logique du premier ordre, il nous semble important d'examiner, même rapidement, les tentatives analogues réalisées dans d'autres cadres formels. La plus séduisante est certainement celle réalisée dans le cadre de la théorie des probabilités.

Nous décrivons ici rapidement les choix fondamentaux qui ont permis l'élaboration de ce modèle extrêmement fructueux (cf. [Rewu 79]).

1.5.1 Notion statistique de test

Considérons un système S susceptible de produire des évènements. L'ensemble de ces évènements éventuels est noté Ω.

Cet ensemble est muni d'une tribu $\mathcal{A}$.

Le système S produit les évènements de Ω

selon une loi de probabilité P (mesurable selon $\mathcal{A}$) inconnue.

Tout ce que nous savons est que P se trouve parmi les éléments d'une famille de $\mathcal{A}$-probabilités sur Ω, $(P_\theta)_{\theta \in \Theta}$. Chacun des P_θ règle donc éventuellement le comportement de S, mais un seul le fait réellement.

Soit $\{\Theta_0, \Theta_1\}$ une partition de Θ. Les membres de la famille $(P_\theta)_{\theta \in \Theta}$ sont donc répartis en deux groupes H_0 et H_1 : $H_0 = \{P_\theta, \theta \in \Theta_1\}$, $H_1 = \{P_\theta, \theta \in \Theta_1\}$.

A la vue d'un événement produit par S, il nous faut décider si $P \in H_0$ ou non.

A chaque événement ω de Ω, l'on associe la décision prise. L'on obtient ainsi une règle de décision δ, application de Ω dans $\{0, 1\}$. C'est donc la fonction indicatrice d'un sous-ensemble W de Ω. W est <u>la région de rejet</u> de l'hypothèse H_0.

1.5.2. Anisotropie

Nous pouvons donc identifier δ et W. La donnée de W (de δ) définit <u>un test de H_0</u>

(contre H_1).

Choisissant de définir le test d'une hypothèse H_0 <u>contre</u> une hypothèse H_1, cette modélisation introduit une notion de test <u>anisotrope</u>, privilégiant H_0 par rapport à H_1.

Ceci conduit à distinguer deux sortes d'erreurs.

Tout d'abord, rejeter H_0 alors qu'elle est vérifiée constitue une <u>erreur de première espèce</u>.

Ensuite, accepter H_0 alors qu'elle n'est pas vérifiée constitue une <u>erreur de seconde espèce</u>.

Une erreur de première espèce est considérée comme plus grave qu'une erreur de seconde espèce. Il est plus grave de rejeter à tort H_0 que de l'accepter à tort.

Cette anisotropie a pour conséquence immédiate que l'on comparera deux tests d'abord selon leur capacité à éviter l'erreur de première espèce, et, seulement en deuxième lieu, selon leur capacité à éviter celle de seconde espèce.

La notion de probabilité permet facilement de transcrire ces intuitions.

Nous définissons donc le <u>niveau</u> d'un test W par :

$$\alpha_W = \sup_{\theta \in \Theta_0} P_\theta(W)$$

C'est donc la probabilité maximale en θ de commettre un erreur de première espèce (rejeter à tort H_0).

Nous définissons aussi la <u>puissance</u> d'un test comme l'application :

$$\Pi_W : \theta \longrightarrow P_\theta(W) \quad \text{définie sur } \Theta_1 .$$

Etant donné deux tests W_1 et W_2 de niveaux inférieurs à α, l'on dira que W_1 est plus puissant que W_2, si

$$\forall \theta \in \Theta_1, \ \Pi_{W_1}(\theta) \geq \Pi_{W_2}(\theta).$$

Autrement dit, W_1 provoque "moins" d'erreurs de seconde espèce que W_2.

1.5.3. Discussion

La notion statistique de test est donc conforme aux intuitions que nous avons dégagées dans les parties précédentes.

Etant donné un problème concret, un système produisant des évènements selon une loi inconnue, l'on construit une <u>représen</u>-

tation abstraite de ce problème, et l'on construit une famille de tests, un jeu de tests (un test pour chaque sous-ensemble W de Ω). L'on se donne ensuite un moyen de comparer ces tests, de déterminer leur qualité.

Plusieurs points retiennent notre attention et inspirerons directement la modélisation de la notion de processus de test que nous proposons au chapitre suivant.

Tout d'abord, la représentation abstraite du problème fait intervenir un certain nombre de connaissances a priori, d'hypothèses extérieures sur la loi P inconnue. Cette loi inconnue est représentée par une famille de lois $(P_\theta)_{\theta \in \Theta}$ parfaitement connues, et donc manipulables formellement. C'est ici la détermination de cette famille de lois éventuelles qui constitue l'intervention d'hypothèses extérieures. Remarquons que le fait que S soit réellement régi par l'une des lois P_θ n'a que finalement peu d'importance, l'important étant que tout se passe comme si cela était.

Ensuite, nous remarquons que les définitions fondamentales sont uniformes en θ.

Elles ne dépendent donc pas de la loi P_{θ_0} réellement agissante pour le système S, mais seulement de la connaissance <u>a priori</u> que l'on possède de cette loi (la famille $(P_\theta)_{\theta \in \Theta}$). Ce point nous semble particulièrement important pour obtenir des définitions applicables en pratique où le θ_0 en cause est inconnu. Il est, semble-t-il, esquissé en [W&O 80], mais non approfondi.

Enfin, nous remarquons que la notion de test est traitée dès l'origine de manière <u>anisotrope</u>, privilégiant la réussite du test ($\theta \in \Theta_0$) par rapport à son échec. Ce choix se reflète dans les définitions qui ordonnent les tests selon leur capacité à éviter l'erreur de première espèce.

1.5.4 Traitement mathématique

La notion statistique de test a depuis longtemps acquis ses lettres de noblesse en donnant lieu à des réalisations pratiques extrêmement efficaces. Ceci n'a été possible que par un développement théorique du modèle sachant utiliser au mieux la puissance de la théorie des probabilités, et cependant rester proche de l'intuition

et des besoins des utilisateurs.

Il est remarquable que, réciproquement, des nécessités techniques aient pu faire évoluer l'intuition de la notion de test, ouvrant la voie à de nouvelles applications auxquelles l'on n'aurait sans doute pas songé tout d'abord.

Par exemple, des considérations techniques montrent que la notion de règle de décision doit être étendue à une application ($\mathcal{A}$-mesurable) de Ω dans $[0,1]$.

Une interprétation intuitive pourrait être la suivante. Si ω est l'événement de Ω produit par S, pour choisir entre "accepter H_0" et "rejeter H_0", l'on réalise une nouvelle épreuve aléatoire qui fait rejeter H_0 avec la probabilité $\delta(\omega)$ (et accepter H_0 avec la probabilité $1-\delta(\omega)$). Si δ est en fait à valeur dans $\{0,1\}$, l'on retrouve l'intuition première.

Cet enrichissement de l'intuition permet de poser des définitions extrêmement élégantes et fructueuses.

<u>définition</u>

On appelle <u>test</u> de H_0 (contre H_1) toute application δ ($\mathcal{A}$-mesurable)

$\delta : \Omega \longrightarrow [0,1]$

On appelle <u>puissance</u> d'un test δ l'application $\Pi\delta$

$$\Pi\delta : \Theta \longrightarrow [0,1]$$
$$\theta \longrightarrow E_\theta(\delta) = \int_\Omega \delta(x)\, dP_\theta(x)$$

On appelle <u>niveau</u> d'un test δ le réel α de $[0,1]$

$$\alpha = \sup_{\theta \in \Theta_0} \Pi\delta(\theta)$$

Cependant, l'enrichissement de l'intuition soulève de nouveaux problèmes de cohérence, cachés lors des étapes précédentes.

Le problème présent est celui du <u>biais</u>. Intuitivement, un "bon" test devrait être tel que si $\theta_0 \in \Theta_0$, alors la conclusion "accepter H_0" devrait être favorisée vis à vis de "rejeter H_0". Réciproquement, si $\theta_0 \in \Theta_1$, ce devrait être le contraire, θ_0 étant la probabilité régissant réellement S.

S'il n'en est pas ainsi, le test nous amènerait, dans certains cas éventuellement, à commettre une erreur plus probablement qu'à ne pas en commettre. Un tel test est dit <u>biaisé</u>.

Il nous faut maintenant retranscrire ceci dans notre formalisme, par une définition uniforme en θ.

<u>définition</u>

Un test δ est dit sans biais, si :

$$\inf_{\theta \in \Theta_1} \Pi\delta(\theta) \geq \sup_{\theta \in \Theta_0} \Pi\delta(\theta) = \alpha .$$

Autrement dit, la probabilité maximum de rejeter <u>à tort</u> H_0 est inférieure à probabilité minimum de rejeter <u>avec raison</u> H_0

La notion de biais joue un rôle extrêmement important dans les développements de la théorie statistique du test (Lemme de Neymann - Pearson).

1.5.5. Conclusion

La notion statistique de test nous offre l'exemple d'une modélisation complète, rigoureuse, aux nombreuses applications pratiques de la notion de test.

Cette modélisation est conforme aux conclusions exposées dans les parties précédentes. Elle nous montre de plus l'influence déterminante que peuvent avoir les exigences techni-

ques sur l'idée intuitive guidant le développement du modèle. Elles permettent à celle-ci de s'enrichir en mettant de nouveaux concepts en évidence.

Une large part de l'élégance de cette approche est due à la puissance de la théorie des probabilités qui la sous-tend. En particulier, celle-ci permet de définir une <u>notion intrinsèque de qualité</u> d'un test (le niveau α), ce que la logique du premier ordre ne nous permet pas. En effet, à la différence des probabilités, c'est un formalisme essentiellement discret et non continu, qui ne se prête donc pas à une telle évaluation quantitative (cf. 9.4.)

1.6. Conclusion

Ce chapitre nous a conduit à étudier, de manière critique, diverses acceptions du mot "test" dans différents domaines.

Le domaine des mathématiques nous a permis de retrouver les propositions développées par GOODENOUGH & GERHART et leurs successeurs (cf. [Whit 81]). Nous y avons décrit aussi une certaine spécificité du test par rapport à la preuve, tout en montrant que ces notions sont étroitement liées.

Le domaine de la vie quotidienne nous a montré que les propriétés sous test peuvent être de diverses formes, dépendant essentiellement de la calculabilité des symboles considérés. Certaines formes sont plus propices à l'élaboration de "bons" jeux de tests que d'autres. Certaines hypothèses, hypothèses de régularité ou d'uniformité, sont alors nécessaires.

Le domaine du logiciel met en évidence les liaisons entre la notion de test, et celles d'oracle de correction, de stratégie et de mesure de qualité.

Le domaine des bases de données nous permet de préciser ces notions dans un cadre un peu plus rigoureux. Nous y trouvons la notion

du jeu de tests. Les notions de coût et de qualité d'un test y sont prépondérantes. Ce domaine met clairement en évidence la spécificité de la notion de test qui est d'introduire le probable là où l'on ne connaissait que le vrai et le faux.

Enfin, l'étude de la notion statistique de test met en évidence l'importance de la phase de représentation abstraite d'un problème concret, et l'importance d'une définition anisotrope de la notion de test privilégiant la réussite du test. Cette modélisation nous montre aussi les interactions profondes s'établissant entre les exigences techniques et l'intuition. La théorie des probabilités permet à ce modèle de proposer une définition intrinsèque de la qualité d'un test pour un problème donné.

A partir des diverses exigences intuitives que nous avons ainsi mises en évidence, nous pouvons maintenant décrire la modélisation mathématique que nous proposons pour la notion de test.

Chapitre 2 : modélisation.

Le chapitre précédent, consacré à l'étude intuitive et empirique de la notion de test, dans les différentes acceptions du mot, nous a permis de dégager un certain nombre d'exigences que notre modélisation devra satisfaire.

Il nous a montré que la notion de test est inséparable de la notion de jeu de tests, ou, plus généralement, de la notion de <u>processus de test</u>, si nous englobons dans cette expression la suite des opérations effectuées lors du test d'une propriété sur un objet, depuis l'examen du problème jusqu'à la conclusion affirmant que l'objet vérifie, ou ne vérifie pas, la propriété.

Il nous a aussi montré que la notion de test est intimement liée à celle de rentabilité, et donc à celles de qualité et de prix. En particulier, notre modélisation devra, entre le "correct" et l'"incorrect", laisser place au "probablement correct", au "suffisamment correct", et introduire une notion de <u>qualité</u> de la correction proposée.

Puisque nous avons en projet l'étude théorique et abstraite de notre modèle, il nous faudra choisir avec soin, et justifier et critiquer explicitement ce choix, l'outil mathématique que nous utiliserons. En fait, il nous semble que l'absence d'une telle critique, d'une telle mise à distance, est l'un des points faibles des théories actuelles du test, résumées par WHITE [Whit 81].

Dans ce chapitre, nous décrivons tout d'abord la notion de processus de test dans sa généralité, précisant particulièrement l'intervention du principe de couplage, qui introduit le probable dans la notion de test. Cette description met en évidence les notions de contexte de test, de restriction conservative de contexte, et de jeu de test acceptable.

Une seconde partie est consacrée au choix de l'outil mathématique nécessaire pour modéliser ces notions, et à quelques principes découlant de ce choix.

La troisième partie élabore les définitions abstraites des notions de contexte et de jeu de tests à partir d'un exemple concret.

2.1. Notion de processus de test

Considérons l'algorithme suivant, tiré d'un exercice de [Reyn 81].

```
RC: begin  y := 0 ; r := x ;
      while  r ≥ 2 * y + 1  do
         begin  y := y+1 ; r = r - (2×y+1)  end
    end
```

Nous voulons tester la propriété suivante

$$\{x \geq 0\}\ RC\ \{y \geq 0,\ y^2 \leq x < (y+1)^2\}$$

Le but de ce paragraphe est de décrire la suite des opérations qui vont nous permettre de construire un test adéquat pour ce problème.

Plus généralement, nous avons un objet P, le plus souvent un programme, sur lequel nous voulons vérifier une propriété A, le plus souvent un couple d'assertions à la HOARE. Si A exprime la correction de P (spécification), nous dirons que nous souhaitons valider P.

P peut être envisagé comme un programme traitant une donnée x, et produisant une valeur y. A est alors une formule, et nous cherchons à tester $\forall x\ A(x, P(x))$.

Une telle propriété n'est en général pas

directement, effectivement calculable, vérifiable, à cause du domaine infini sur lequel x varie. Par contre, l'on peut le plus souvent vérifier que pour une valeur x_0, $A(x_0, P(x_0))$ (propriété spatiale universelle, cf. 1.2.)

La notion de processus de test se décompose en trois phases :

- <u>représentation abstraite</u> de l'objet et de la propriété, dans un cadre formel convenable, pour construire un <u>contexte de test</u>.
- <u>construction</u> d'un <u>jeu de tests</u> (abstrait) sur un contexte éventuellement restreint. Ce jeu de tests doit avoir un certain nombre de "bonnes" propriétés.
- <u>application</u> du jeu de tests au problème concret par le choix d'un test, la vérification de ce test sur P, et la conclusion par invocation du <u>principe de couplage</u>.

$$\begin{tikzcd}
\text{niveau concret} & P, A \arrow[rr, dashed, "\text{conclusion}"] \arrow[d, "\text{représentation}"'] & & \text{test } T \\
\text{niveau abstrait} & \text{contexte } \mathcal{C} \arrow[rr, "\text{construction}"'] & & \text{jeu de tests } \mathcal{C} \arrow[u, "\text{application}"']
\end{tikzcd}$$

2.1.1 représentation

Le problème se décompose en deux parties:
représentation de l'objet, et
représentation de la propriété sous test.

2.1.1.1 représentation de l'objet

Dans la pratique, nous ne connaissons qu'imparfaitement l'objet considéré. Par exemple, si P est un algorithme, nous ne connaîtrons, en un temps fini, son comportement que pour un nombre fini de valeurs. Si P est un système informatique, nous ne connaissons que la forme externe, en langage de haut niveau, du programme exécuté par la machine, et non la suite de micro-instructions conduisant réellement son comportement.

Nous sommes donc amené à donner, non pas une, mais des représentations abstraites potentielles de P, plus ou moins nombreuses suivant notre degré de connaissance de cet objet. Nous représentons donc un objet connu imparfaitement par une famille d'objets abstraits parfaitement connus, que nous noterons (F_0).

Si P est le programme RC, (F_0)

sera par exemple la famille des fonctions "proches" de la fonction "racine carrée entière". Plus généralement (Fo) sera une famille de fonction sur le domaine de P éventuellement réalisées par P. Elle contiendra donc, entre autres, la fonction réalisée par P, mais il ne nous est pas donné de la connaître plus avant.

Il sera bien sûr souhaitable de manipuler une famille (Fo) la plus restreinte possible.

2.1.1.2 représentation de la propriété

Pour que la représentation précédente soit utilisable, il nous faut être capable de traduire la propriété A en une autre propriété, notée aussi A, concernant les membres de la famille (Fo).

Il faut de plus que cette traduction soit cohérente, en ce sens que si A est vérifiée par P, il faut qu'elle le soit aussi par le Fo représentant réellement P, et réciproquement.

Dans notre exemple, ayant choisi le cadre de la logique du premier ordre, la propriété

$$\{x \geq 0\}\ P\ \{y \geq 0,\ y^2 \leq x < (y+1)^2\}$$

se traduira par

$$\forall x \, [\, x \geq 0 \rightarrow F(x) \geq 0 \wedge F(x)^2 \leq x < (F(x)+1)^2 \,]$$

où F est la fonction réalisée par P, et membre de (F_0)

2.1.1.3 Conclusion

Nous avons donc décrit la première partie du diagramme

$$\begin{array}{c} \langle P, A \rangle \\ \Big\downarrow \text{ représentation.} \\ \langle (F_0), A \rangle \end{array}$$

Nous appellerons $\langle (F_0), A \rangle$ un <u>contexte de test</u>.

2.1.2 construction

Le problème est maintenant de construire un "bon" jeu de tests sur le contexte $\langle (F_0), A \rangle$.

Puisque nous ne savons pas quel F_0 sera réellement manipulé lors de l'application concrète du test choisi, le jeu de tests construit, ou plus généralement toutes les pro-

priétés introduites devront être <u>uniformes</u> par rapport à la famille (F_0)

2.1.2.1 expérience, test et jeu de tests.

Nous avons étudié au chapitre 1 le sens du mot "tester". Nous sommes donc conduit à définir un certain nombre de propriétés élémentaires, vérifiables en un temps sur un F_0 donné, quel que soit ce F_0. Une telle propriété élémentaire sera dite une <u>expérience</u>. Une expérience typique est de demander la valeur d'une fonction (calculable) en un point. Ces expériences sont telles, bien sûr, que leur traduites soient effectivement réalisables sur $\mathcal{P}$.

Un <u>test</u> T est un ensemble fini d'expériences.

Un <u>jeu de tests</u> est une famille ordonnée de tests $(T_n)_{n \in \mathbb{N}}$

Etant donnée un membre F_0 de (F_0), une expérience peut soit réussir, soit échouer; la propriété élémentaire est, ou n'est pas vérifiée par F_0. F_0 passe le test T si toutes les expériences du test T réussissent sur F_0.

En fait, nous imposerons une condition de cohérence supplémentaire sur la famille $(T_n)_{n \in \mathbb{N}}$ d'un jeu de tests : la <u>fiabilité projective</u>

(cf. 2.3.5. et 3.1.). Etant donné T_0, si T_0 passe T_n, alors T_0 passe T_m, pour tout $m \leq n$.

2.1.2.2. Jeu de tests acceptable

Le jeu de tests que nous construisons doit être capable de nous apporter des informations sur la validité de la propriété A sur P. Il doit donc posséder certaines propriétés : la validité asymptotique, et l'absence de biais (cf. 3.2. et 3.3.).

Un jeu de tests est asymptotiquement valide si pour tout T_0, si T_0 passe tous les T_n, alors T_0 vérifie A.

Un jeu de tests est sans biais si pour tout T_0, si T_0 vérifie A, alors T_0 passe tous les T_n.

Un tel jeu de tests est alors dit acceptable (logiquement, cf. 3.4.) : pour tout T_0, T_0 vérifie A ssi T_0 passe tous les T_n.

2.1.2.3 Restriction de contexte

Malheureusement les jeux de tests que nous obtiendrons ne seront en général pas acceptable sur le contexte $\langle (T_0), A \rangle$. Ils ne le

seront que sur un sous-contexte $<(F), A>$ avec $(F_0) \supseteq (F)$. Nous opérerons donc une <u>restriction de contexte</u>.

Remarquons que, puisque nous ne connaissons pas le F_0 qui représente réellement P, nous ne sommes plus capables d'affirmer, avec certitude, après une restriction de contexte, s'il existe toujours un membre de la famille (F) représentant P. C'est pourquoi nous n'opérerons que des restrictions de contexte <u>conservatives</u> (cf. 5.1.)

Une restriction de contexte de $<(F_0), A>$ à $<(F), A>$ (donc telle que $(F_0) \supseteq (F)$) sera dite conservative si tout F_0 vérifiant A appartient à (F). La restriction conserve donc les représentants potentiels de P si P vérifie A.

2.1.2.4 Conclusion

Nous avons donc décrit la seconde partie de notre diagramme

$$\text{contexte } <(F_0), A> \xrightarrow{\text{construction}} \text{jeu de tests } (T_n)_{n \in \mathbb{N}} \text{ acceptable sur le contexte } <(F), A>$$

le contexte $<(F), A>$ étant une restriction conservative du contexte $<(F_0), A>$.

2.1.3 Application

En retraduisant chacun des T_n en une suite (finie) d'expériences effectives sur P, nous sommes maintenant amené à choisir un test, et à le réaliser sur P. Dans notre exemple, T_n serait, probablement, la suite d'expériences suivantes :

" Exécuter l'algorithme RC pour $x = 0$, $x = 1, \ldots, x = n$, et dans chaque cas vérifier la validité des assertions "

Il est bien évident qu'une telle expérience, si elle réussit, ne fait qu'apporter une présomption de validité pour la propriété $A = \{x \geq 0\}$ RC $\{y \geq 0,\ y^2 \leq x < (y+1)^2\}$, plus ou moins grande suivant la valeur de n, mais croissante avec n.

Supposons donc que pour chaque n, nous soyons capable de définir un indice de qualité. Cet indice mesurera, intuitivement, la certitude avec laquelle nous pourrons affirmer par la suite que P vérifie A, alors que nous savons seulement que P vérifie T_n. Cet indice dépendra donc, dans une large mesure, de la

manière dont nous avons construit $(T_n)_{n\in\mathbb{N}^*}$, mais aussi de l'utilisation future du fait que P vérifie A. En particulier, si ce fait est crucial, l'indice croîtra lentement avec n. S'il est presque indifférent, il croîtra plus vite. Ce point est donc essentiellement subjectif.

A chaque n, nous pouvons aussi associer le prix de l'expérience T_n (temps de calcul, manque à gagner dans l'exploitation commerciale).

Au vu de la somme dont nous disposons, nous choisissons alors k tel que T_k soit de prix abordable, et de rapport qualité / prix (rentabilité) optimal. Nous examinons alors si P vérifie T_k.

Si P vérifie T_k, nous <u>concluons</u> que P vérifie A.

Si P ne vérifie pas T_k, nous <u>concluons</u> que P ne vérifie pas A.

Nous avons donc construit la troisième partie de notre diagramme :

$$\begin{tikzcd}
\langle P, A\rangle \arrow[r, dashed] & T_k \arrow[r, dashed] & \text{conclusion} \\
& \text{application} \arrow[u] & \\
& \text{jeu de tests } (T_n)_{n\in\mathbb{N}^*} &
\end{tikzcd}$$

2.1.4 Principe de couplage

Au paragraphe précédent, nous avons décrit la conclusion du processus de test. Du fait que P vérifie Tk, nous avons proposé de conclure le fait que P vérifie A.

Il est bien clair que, en toute rigueur, il n'y a aucune raison que les deux faits soient liés. C'est précisément dans l'action de les lier ainsi que s'introduit la notion de probable, qui est la spécificité de la notion de test par rapport à celle de preuve.

2.1.4.1 Probabilité et certitude

Si nous avons écrit "probable", et non "aléatoire", c'est que notre conclusion, pour arbitraire qu'elle soit, peut être étayée par un certain nombre de certitudes.

Supposons que P vérifie tous les Tn du jeu de tests construit sur le contexte restreint (conservativement) <(F), A>.

Alors supposons que P vérifie A. La représentation abstraite de P, puisque nous avons réalisé une restriction conservative, est donc membre de (F), et vérifie tous les Tn.

Nous pouvons en conclure que cette représentation vérifie A, et donc que P vérifie A. En toute rigueur, nous n'avons rien démontré, mais nous avons cependant mis en évidence une certaine cohérence.

Supposons que P ne vérifie pas A. Si la représentation abstraite de P était dans (F), alors elle ne vérifierait pas tous les T_n ; contradiction ! Elle n'est donc pas dans (F).

Supposons maintenant que P ne vérifie pas tous les T_n (un cas particulier est celui où P ne vérifie pas T_k).

Si P vérifiait A, la représentation abstraite de P serait dans (F), et, le jeu de tests étant acceptable, elle vérifierait tous les T_n ; contradiction ! Donc P ne vérifie pas A.

Conclure que P ne vérifie pas A à partir du fait que P ne vérifie pas T_k est donc <u>rigoureusement fondé</u>. Si nous reprenons les notions du test statistique en posant H_0 = "P vérifie A" et H_1 = "P ne vérifie pas A", nous remarquons que nous ne refusons H_0 qu'<u>avec certitude</u>.

L'erreur de première espèce (cf. 1.5.) considérée comme plus grave que celle de seconde espèce est donc exclue. Notre démarche est donc tout à fait cohérente avec celle proposée par ce formalisme.

2.1.4.2 Enoncé

Conclure que P vérifie A à partir du fait que P vérifie Th n'est pas, a contrario, rigoureusement fondé.

Cette action l'est si, en fait,

P vérifie tous les T_n

la représentation de P est dans (F)

La qualité de notre conclusion est donc exactement la certitude avec laquelle nous pouvons induire que des résultats rigoureusement valables dans ce cas restreint le restent dans un cas un peu plus général.

Le principe de couplage peut s'énoncer ainsi

<u>principe de couplage</u> :

Appliquer un raisonnement rigoureux à des données vérifiant presque les hypothèses conduit à un résultat probablement correct.

Dans notre situation, nous considérons que si la qualité de T_k est satisfaisante ("k assez grand"), et si la représentation de I est probablement dans (F) (I vérifie probablement les hypothèses d'uniformité), alors nous sommes dans les conditions d'application du principe, et <u>tout se passe comme si</u> I vérifiait tous les T_n, la représentation de I étant dans (F).

2.1.4.3 Conclusion

Par l'introduction du principe de couplage, nous avons introduit le "probable" dans la notion de processus de test.

C'est la spécificité de la notion de test que d'assurer seulement une présomption de certitude, de plus ou moins grande qualité. Mais, en général, l'on ne dispose d'aucune indication sur cette qualité.

Notre modélisation, en localisant très précisément l'application du principe de couplage, et en dégageant ce qui est "rigoureusement certain" de ce qui n'est que "probablement certain", nous donne les moyens de mesurer, quoique subjectivement encore,

la qualité de la conclusion d'un test.

2.1.5 Conclusion

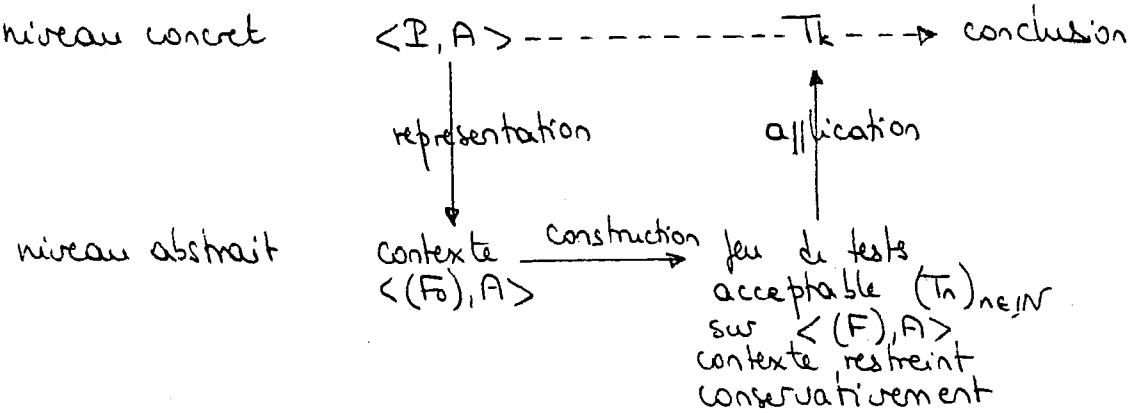
Cette première partie nous a permis de présenter en détail la notion de processus de test sur laquelle est fondé ce travail.

Le processus de test d'une propriété A sur un objet <u>incomplètement connu</u> P se décompose en trois phases.

La phase de représentation abstraite remplace l'objet P incomplètement connu par une famille de représentations potentielles (F_0). Cette phase produit un contexte de test.

La phase de construction produit un jeu de tests acceptable sur un contexte éventuellement conservativement restreint.

La phase d'application est constituée par le choix d'un test T_k convenable, et sa vérification sur P.



Enfin, nous avons décrit le principe de couplage, dont l'application permet de conclure, si P vérifie T_k, que P vérifie A comme une "certitude probable".

2.2 Choix d'un outil mathématique.

La première partie a dégagé, conformément aux intuitions présentées au chapitre précédent, la notion de processus de test d'une propriété sur un objet.

En pratique, l'objet sera un système informatique exécutant un certain programme objet. La propriété sera la spécification du programme source dont il est issu, ou plus généralement, la donnée d'une assertion d'entrée et d'une assertion de sortie, à la HOARE.

La notion de processus comprend une phase de construction d'un jeu de tests sur un contexte. Cette construction, pour être rigoureuse, doit se faire dans un cadre formel, mathématique bien délimité. Il doit permettre de représenter le comportement global de l'objet, ainsi que la propriété considérée. Il doit aussi permettre d'exprimer qu'une propriété est vérifiée par les éléments d'une famille de représentations

Cette partie est consacrée au choix d'un tel cadre, qui sera le formalisme de la logique du premier ordre. Les critères et les

conséquences de ce choix seront examinées en détail.

Une introduction succinte à la logique du premier ordre a été placée en annexe 3 du présent travail.

2.2.1. Critères de choix

Examinons les considérations techniques qui peuvent nous guider dans notre choix.

Notre cadre formel doit, tout d'abord, être suffisamment proche du cadre intuitif qui a servi de base à ce travail. Il est particulièrement lié aux notions de programme, de validation, de correction, de spécification, d'erreur [Boug 82]. Y interviennent aussi les notions de coût, de qualité, de critère de sélection, de domaine (cf. chapitre 1). Une distinction y est faite entre propriétés calculables et transcendantes.

Par ailleurs, nous voulons être capable de formuler des propriétés précises, sans ambiguïté, abstraites, dans un langage mathématique clairement défini. De plus, nous voulons n'utiliser que des notions mathématiques classiques, abordables, "culturellement assimilées", largement diffusées et

utilisées, ceci pour permettre un traitement théorique efficace et fructueux de notre modèle. Nous excluons a priori toute référence à des notions trop neuves ou trop peu explorées.

Ce choix ne peut être, en grande partie, qu'arbitraire. Nous sommes d'autre part conscient de ce que l'outil mathématique choisi détermine largement les résultats obtenus. Ces résultats seront de natures très différentes si nous les exprimons dans le cadre de la logique du premier ordre ou dans celui de la théorie des probabilités. Les résultats et les exemples présentés dans cette thèse devront donc toujours être examinés en relation avec les contraintes liées à la logique du premier ordre. Citons tout particulièrement l'utilisation conjointe de H et de (S), la notion de théorie logique, la dualité entre propriétés logiques et propriétés formelles, le problème des hypothèses d'uniformité et de régularité, nécessaires en théorie, mais totalement virtuelles en pratique, l'impossibilité de (bien) tester des propriétés non purement universelles (cf. 3.4.3), etc.

2.2.2. Choix de la logique du premier ordre

Considérant les exigences présentées plus haut, deux langages mathématiques, parmi ceux que nous connaissons, restent en lice. L'un est celui des probabilités, sous-tendu par la théorie de l'intégrale de Lebesgue. L'autre est celui de la logique, et plus spécialement de la logique (égalitaire) du premier ordre.

Le test statistique, depuis de nombreuses années, a été largement étudié et diffusé, et a donné lieu à des résultats extrêmement élégants et intéressants dans de nombreux domaines. Cependant, cette théorie semble peu adaptée à la notion de programme algorithmique, et plus généralement, semble disjointe des notions de calculabilité et de preuve. La notion de test à laquelle elle conduit est donc difficilement exploitable dans notre cas, puisque, comme le montre [Boug 82], la notion de test en informatique a toujours été conçue comme complémentaire de celle de preuve.

La logique du premier ordre est affirmée, pour la plupart des auteurs cités en [Boug 82], et notamment par GOODENOUGH & GERHART [G&G 75], comme le langage "naturel" de la validation de programmes par test. Une propriété

est alors une formule (close). Ce formalisme est de plus proche de celui des types abstraits algébriques (de nombreuses références sont fournies par [Gaud 79], [Gaud 80], [Bido 81]), dans lequel peuvent se décrire la plupart des objets informatiques. Les notions de preuve et de calculabilité s'expriment aisément et puissamment en logique [Shoe 67]. Le problème du coût d'évaluation d'une formule a été étudié, notamment dans le cas des bases de données [Nico 79], [Demo 82]. La notion de processus, par l'introduction du principe de couplage dans la conclusion du test, nous dispense d'exprimer le "probable" dans notre formalisme.

Malheureusement, au terme de ce travail, nous ne serons pas capable, dans le cadre de ce formalisme, de donner une définition de la notion de <u>qualité d'un test</u>. L'expression de cette notion typiquement statistique dans le cadre que nous avons tracé serait sans doute l'objet de recherches extrêmement intéressantes et fructueuses, tant sur le plan théorique que pratique (cf. 9.4.)

2.2.3. Conséquences.

La logique (égalitaire) du premier ordre nous fournit un outil mathématique qui va nous permettre de développer un formalisme rigoureux autour de la notion de test. Cet outil est déjà bien connu, bien dominé, largement utilisé ; nous avons à notre disposition de nombreux théorèmes et contre-exemples, des techniques de démonstration bien délimitées, et un jeu de définitions parmi lesquelles nous pouvons puiser celles adaptées à nos besoins (cf. annexe 3)

Le fait de pouvoir disposer d'une telle base mathématique nous paraît fondamental pour ce travail. Cependant, il implique de nombreuses contraintes.

Tout d'abord, cet outil se fonde sur la notion de théorie, notion sans doute trop restrictive alors qu'un ensemble quelconque d'axiomes nous aurait suffi. Mais, en fait, l'emploi que nous ferons de cette notion nous permettra de ne nous préoccuper que de la partie non-logique des théories considérées.

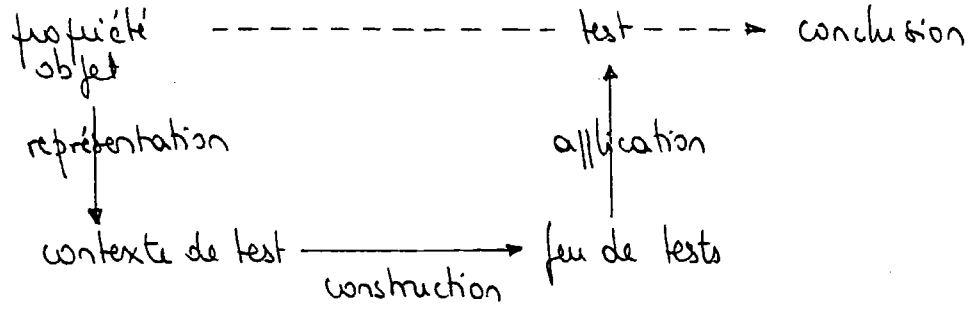
Ensuite, la dualité entre propriétés logiques et formelles, fondamentale dans cet outil, devra être conservée dans nos défini-

tions si nous voulons pouvoir exploiter toute la subtilité de ce formalisme. Cela nous conduira parfois à des énoncés alourdis d'adjectifs et d'adverbes, mais plus précis que l'intuition seule ne l'aurait peut-être exigé. A terme, cela nous permettra de distinguer la notion théorique de test, plutôt du domaine formel, et ses applications pratiques, plutôt du domaine logique. L'intuition n'aurait peut-être pas su tracer une séparation aussi nette et aussi fructueuse.

Enfin, cette approche privilégiera les applications issues du formalisme des types abstraits algébriques, qui peut être vu comme un cas particulier de la logique (égalitaire) du premier ordre. Il est d'ailleurs remarquable que la forme des axiomes des types abstraits (axiomes purement universels) soit la seule où notre formalisme s'applique directement (existence d'un jeu de tests formellement acceptable, cf. 3.4.3). La plupart de nos exemples seront d'ailleurs de cette forme (cf. chapitres 7 et 8, 9.3.)

2.3. Contexte et jeu

Etant maintenant en possession d'un outil mathématique bien défini, nous pouvons continuer la modélisation entreprise dans la première partie. Il nous faut donc définir rigoureusement ce que nous entendons par contexte de test et jeu de tests



Etant donné le rapport étroit entre logique du premier ordre et types abstraits algébriques noté en 2.2.3, notre démarche pour représenter un objet (programme) par une famille de structures logiques sera très proche de celle proposée en [Gaud 80]. A un système informatique incomplètement connu, nous associons une famille de structures, qui peuvent être vues comme autant de Σ-algèbres représentant les comportements externes (boîte noire) éventuels du système, ($\mathcal{S}$). La classe de ces Σ-algèbres est restreinte au moyen d'axiomes (formules du premier ordre), qui

représentent les hypothèses formelles sur le comportement du système, H.

Considérons donc un objet P et une propriété A. P est, par exemple, le programme RC présenté au début du chapitre. A est alors la propriété

$$\{x \geq 0\}\ RC\ \{y \geq 0,\ y^2 \leq x < (y+1)^2\}.$$

Nous supposons que l'objet P peut être modélisé par une boîte noire $y = F(x)$. En particulier, cela suppose que nous sachions a priori (ou, en tout cas, que nous fassions comme si) <u>P s'arrête pour toute valeur x</u> en un temps fini, en fournissant une valeur y, x variant sur le domaine de la fonction modélisant P (ici, $\mathbb{N}$ suffit).

2.3.1 Langage, individus et structures

P peut donc être abstrait comme application de $\mathbb{N}$ dans $\mathbb{N}$. Dans le langage des types abstraits, P induit sur $\mathbb{N}$ une structure de Σ-algèbre (avec $\Sigma = \langle F \rangle$). De même, dans notre formalisme, nous nous donnons un langage (égalitaire) du premier ordre, dont le seul symbole non-logique est F, symbole

fonctionnel unaire (nous noterons $L = \{F\}$).
Nous nous donnons aussi un ensemble de base $S = \mathbb{N}$, $\mathcal{P}$ détermine alors une L-structure $\mathcal{S}$ sur S, c'est à dire une application $F_{\mathcal{S}}$ de S dans S (cf. annexe 3)

Comme nous l'avons dit, $\mathcal{P}$ n'est en général pas complètement connu. Cependant, quelles que soient les nuances de son comportement, il est légitime de poser que $\mathcal{P}$ est un programme sur le domaine des $x \geq 0$ au moins, s'arrêtant pour toutes ces valeurs (par hypothèse) avec une valeur $y \geq 0$. La famille des représentations abstraites de $\mathcal{P}$ sera donc une famille $(\mathcal{S})$ de L-structures <u>sur le même ensemble de base S</u>, notée $\langle L, S, (\mathcal{S}) \rangle$

Il nous faut maintenant exprimer dans ce cadre formel la propriété A. Intuitivement, A s'écrit

$$x \geq 0 \rightarrow F(x) \geq 0 \wedge F(x)^2 \leq x < (F(x)+1)^2,$$

ou plus correctement

$$\forall x \, [\, x \geq 0 \rightarrow F(x) \geq 0 \wedge F(x)^2 \leq x < (F(x)+1)^2 \,]$$

Le langage $L = \{F\}$ est donc trop restreint pour exprimer cette propriété. Il nous faut considérer une <u>extension</u> de ce langage, contenant les symboles non-logiques $\_ \geq \_$, 0, $\_^2$

$-\leq-<-$, $-+-$, 1 , au moins. Soit L' ce langage. Chacune des L-structures définies précédemment peut être étendue en une L'-structure, en donnant à ces symboles leur interprétation habituelle. En particulier, ces structures vérifient, par exemple, les formules suivantes:

$$\forall x\ [\ x \geq 0]$$

$$0+0=\ldots\ 0\ !$$

$$0+1=1$$

$$\forall x \forall y\ [x+y=y+x]$$

$$\forall x \forall y \forall z\ [x \leq y < z \leftrightarrow y \geq x \wedge z \geq y \wedge \neg\, z=y]$$

et bien d'autres formules encore.

Ces structures vérifient aussi des formules que nous pouvons énoncer intuitivement ainsi:

$$2+3=5$$

$$\neg\ 3 \geq 5 \text{ , par exemple.}$$

Pour pouvoir énoncer de telles formules, nous pouvons étendre canoniquement (cf. annexe 3) chaque L'-structure $\mathcal{S}$ en une $L'(s)$-structure. Puisque toutes les structures sont construites sur le même ensemble de base, nous parlerons de $L'(s)$-structures, et noterons $\langle L', s, (\mathcal{S})\rangle$.

2.3.2 Structures et hypothèses

Toutes les $L'(s)$-structures ne représentent pas un comportement éventuel plausible du

programme sous test. Nous ne considérons en fait qu'une famille restreinte de $L'(S)$-structures.

Cette famille est définie par la connaissance a priori que nous avons du programme (cf. 1.5.1). Cette connaissance peut s'exprimer de manière logique, par des conditions sur les applications et prédicats. Par exemple, nous nous restreindrons aux structures $\mathcal{S}$ telles que $F_{\mathcal{S}}$ soit calculable.

Mais, à côté de ces méta-conditions, en général inexprimables dans le langage, nous pouvons poser des hypothèses formelles. Par exemple nous ne considérons que les $\mathcal{S}$ telles que $\mathcal{S} \models \forall x\, [x \geqslant 0]$, ce qui restreint les choix de $\geqslant_{\mathcal{S}}$ et $0_{\mathcal{S}}$

Cette démarche est conforme à celle qui consiste, parmi les Σ-algèbres, à ne se restreindre qu'aux Σ, E-algèbres <u>premières</u>, où E est un système de Σ-équations.

Ici nous sommes donc conduit à considérer des hypothèses constituées de $L'(S)$-formules du premier ordre. Selon l'esprit de notre travail, nous nous restreindrons au cas où cet ensemble de formules est une $L'(S)$-théorie (cf. annexe 3). Cette clause est en fait virtuelle en pratique, puisque seule la partie non-logi.

que de la théorie est nécessaire dans la plupart des cas. Nous noterons cette théorie H, ou plutôt H_0, pour marquer qu'elle correspond aux hypothèses initiales de construction (cf. 5.3.).

Dans notre exemple, H est la L'(S)-théorie contenant, comme axiomes non-logiques, les formules (closes) exprimant que les symboles ont leur signification habituelle: $\forall x \quad x \geq 0$

$\forall x \forall y \ (x+y = y+x)$

$2 \leq 3 < 5$

etc.

Il y a, en général, une infinité de formules, éventuellement toutes celles vérifiées par toutes les structures de $(\mathcal{S})$.

Par notre choix, pour tout $\mathcal{S}$ de $(\mathcal{S})$, $\mathcal{S} \models H$, $\mathcal{S}$ est modèle de H. Mais $(\mathcal{S})$ ne contient pas, en général, tous les S-modèles de H, et encore moins tous les modèles de H (cf. annexe 3, théorème de non-catégoricité).

2.3.3. Propriété sous test et test

La propriété sous test, A, est donc représentée par une formule du premier ordre.

$$\forall x \ [x \geq 0 \rightarrow F(x) \geq 0 \wedge F(x)^2 \leq x < (F(x)+1)^2 \ ]$$

Une structure $\mathcal{S}$ vérifie A si elle valide A ; $\mathcal{S} \models A$. Si $\mathcal{S}$ valide A, $\mathcal{S}$ valide la théorie

(la L'(S)-théorie) ayant A pour seul axiome non-logique (et notée aussi A).

La représentation d'une propriété à tester sera donc une théorie. Mais il n'y a aucune raison de se restreindre au cas où celle-ci ne compte qu'un seul axiome non-logique. Il nous sera utile (cf. chapitres 5, 6 et 7) de pouvoir tester conjointement une infinité de propriétés. Si $\mathcal{S}$ représente P, $\mathcal{S} \models A$ si et seulement si P vérifie A.

Une expérience E sera simplement une formule du premier ordre <u>effectivement vérifiable sur</u> ($\mathcal{S}$): étant donné une structure $\mathcal{S}$ de ($\mathcal{S}$), il est possible, en un temps fini, de décider si $\mathcal{S} \models E$ ou non, si l'expérience réussit ou échoue. S étant infini en général, une expérience sera une formule sans variable telle que l'interprétation de chaque symbole soit calculable, et ceci pour toute structure de ($\mathcal{S}$).

Un test T est une théorie n'ayant qu'un <u>nombre fini d'axiomes logiques</u>, ceux-ci étant des expériences. Etant donné une structure $\mathcal{S}$, $\mathcal{S} \models T$ si et seulement si $\mathcal{S}$ valide chacune de ses expériences. Il est donc possible de décider en un temps fini, et ce pour toute structure $\mathcal{S}$, si $\mathcal{S} \models T$ ou non, si $\mathcal{S}$ passe ou

ne passe pas le test T.

2.3.4 Définition : contexte de test

Nous sommes maintenant en mesure de définir rigoureusement les notions que nous avons élaborées au cours de cette partie

<u>définition</u> : contexte de test

Un contexte de test $\mathcal{C}$ est la donnée de
un langage du premier ordre L (égalitaire),
un ensemble S,
une famille de L(S)-structures (sur S), $(\mathcal{S})$
une L(S)-théorie A

Ce que l'on note $\mathcal{C} = \langle L, S, (\mathcal{S}), A \rangle$.

En général, nous manipulerons conjointement une L(S)-théorie H telle que toute structure $\mathcal{S}$ de $(\mathcal{S})$ soit modèle de H. Nous parlerons alors de H-contexte.

A sera appelée théorie sous test, et H théorie d'hypothèses.

Résumons le contexte de test déduit de notre exemple.

$L = \{ F ; \_ \geq \_ ; 0 ; \_^2 ; \_ \leq \_ < \_ ; \_ + \_ ; 1 \}$

$S = \mathbb{N}$

$A = \{ \forall x \, [x \geq 0 \rightarrow F(x) \geq 0 \wedge F(x)^2 \leq x < (F(x)+1)^2] \}$

$(\mathcal{S})$ = l'ensemble des $L(S)$-structures (sur S), telles que $F_{\mathcal{S}}$ soit calculable, et que les autres symboles aient leur signification habituelle.

La notion de contexte est étudiée au chapitre 5.

Remarquons au passage que la famille de toutes les $L(S)$-structures (sur S) est bien un ensemble.

2.3.5 Définition : test et jeu de tests.

Reprenons les définitions proposées en 2.3.3.

<u>définition</u> : expérience

Soit $\mathcal{C} = \langle L, S, (\mathcal{S}), A \rangle$ un contexte de test.
Une expérience E sur $\mathcal{C}$ est une $L(S)$-formule (close) $\mathcal{C}$-effectivement vérifiable

<u>définition</u> : vérifiabilité effective

Soit $\mathcal{C} = \langle L, S, (\mathcal{S}), A \rangle$ un contexte de test.
Une formule de $L(S)$ est dite $\mathcal{C}$-effectivement vérifiable si
elle ne contient pas de variable (ni de quantificateur)
l'interprétation des symboles non-logiques y apparaissant est calculable, pour chacune des structures de $(\mathcal{S})$.

Cette propriété ne dépend donc que de L, S et $(\mathcal{S})$, et non de A.

<u>définition</u> : test

Soit $\mathcal{C} = \langle L, S, (\mathcal{S}), A \rangle$ un contexte de test.
Un test T sur $\mathcal{C}$ est une $L(S)$-théorie ne contenant qu'un nombre fini d'axiomes non-logiques, chacun d'eux étant une $\mathcal{C}$-expérience

Notre définition prend donc bien en compte le difficile problème de l'oracle soulevé par WEYUKER [Weyu 80]

Dans le cas de notre exemple, en abrégeant A en $\forall x\, \phi(x)$, un test intéressant sera, par exemple, la théorie $T_k = \{ \phi(0) \wedge \dots \wedge \phi(k) \}$

Remarquons que si $E_1, \dots, E_n$ sont des $\mathcal{C}$-expériences, $E_1 \wedge \dots \wedge E_n$ aussi. Le lecteur pourra donc toujours concevoir un test comme un théorie n'ayant qu'un seul axiome non-logique celui-ci étant une expérience.
Test et expérience sont donc deux notions virtuellement identiques.

Venons-en à la définition d'un jeu de tests.

<u>définition</u> : jeu de tests

Soit $\mathcal{C} = \langle L, S, (\mathcal{S}), A \rangle$ un contexte de test.
Un jeu de tests $\mathcal{T}$ sur $\mathcal{C}$ est la donnée de

une $L(S)$-théorie H
telle que toute structure de $(\mathcal{S})$ soit
modèle de H
une famille $(T_n)_{n \in \mathbb{N}}$ de tests sur $\mathcal{S}$
telle que pour tout $n \in \mathbb{N}$
$H \sqcup T_{n+1} \vdash T_n$

D'après la définition, $\mathcal{S}$ est alors un H-contexte. La propriété à laquelle est soumise $(T_n)_{n \in \mathbb{N}}$ est étudiée en 3.1. sous le nom de fiabilité projective formelle.

2.3.6 Discussion

Les définitions proposées en 2.3.5. appellent quelques commentaires.

Le chapitre premier a clairement montré l'importance de la notion de jeu de tests. Faire un test n'a de sens que si l'on a le choix du test [G&G 75]. Il nous a semblé important d'introduire une hiérarchie dans ce choix, et nous avons montré que cette intuition s'exprime par :

pour toute structure $\mathcal{S}$ de $(\mathcal{S})$,
si $\mathcal{S} \models T_n$ alors $\mathcal{S} \models T_{n'}$, $n \geq n'$

La famille (T_n) doit donc être ordonnée.

Pour tous les cas concrets que nous avons rencontrés [Boug 82] montrent qu'il est suffisant, empiriquement, de considérer une famille dénombrable, totalement ordonnée. Les exemples traités aux chapitres 6 et 7 semblent le confirmer. Nous avons donc indexé notre famille par $\mathbb{N}$. Cependant, nous n'excluons pas, a priori, des ordres plus complexes, partiels et non-dénombrables, peut-être intéressants sur le plan théorique. Ils ne seront pas considérés ici.

La condition "$\forall \mathcal{S} \in (\mathcal{S})$, si $\mathcal{S} \models T_{n+1}$ alors $\mathcal{S} \models T_n$" aurait donc suffit à rendre compte de notre intuition. Cependant, nous souhaitons, à terme, définir une notion de "bon" jeu de test faisant intervenir $(\mathcal{S})$ le moins possible (cf. chapitre 3, spécialement 3.4.4). Une telle propriété, n'étant pas transposable formellement (le théorème de complétude n'est évidemment pas applicable, puisque $(\mathcal{S})$ ne contient qu'une infime partie des modèles de T_{n+1}), elle ne nous serait donc d'aucune utilité.

Nous serions donc tenté de poser

$\forall n \in \mathbb{N} \quad T_{n+1} \vdash T_n$.

Mais, pour reprendre notre exemple, si l'on sait a priori que $\forall x\, [\phi(x+1) \rightarrow \phi(x)]$, il est bien évident que $\{\phi(3)\}$ est un

test meilleur que $\{\phi(2)\}$, quoique $\phi(3) \not\vdash \phi(2)$.

Nous sommes donc amené à prendre

$\forall n \in \mathbb{N} \quad H \sqcup T_{n+1} \vdash H \sqcup T_n$

ou, de manière équivalente

$\forall n \in \mathbb{N} \quad H \sqcup T_{n+1} \vdash T_n$ (puisque $H \vdash H$).

Cette condition est "un petit peu" plus contraignante que la première, d'autant moins que H est plus précise. Elle est <u>purement</u> formelle, ne dépendant de $(\mathcal{S})$ que par l'intermédiaire de H.

Dans la suite, nous dirons que

$\forall \mathcal{S} \in (\mathcal{S}) \quad \forall n \in \mathbb{N}$ si $\mathcal{S} \models T_{n+1}$ alors $\mathcal{S} \models T_n$

et $\forall n \in \mathbb{N} \quad H \sqcup T_{n+1} \vdash T_n$

sont des <u>conditions logiques et formelles duales</u>. L'expression formelle sera, en général, plus contraignante que sa duale logique. Cette dualité tient une grande place dans notre travail.

2.3.7 Conclusion

Dans cette partie, nous avons construit, justifié et énoncé les définitions qui fondent notre travail : contexte de test et jeu de tests, dans le cadre rigoureux de la logique du premier ordre. Nous pouvons reprendre le diagramme construit dans la première

partie, modélisant le processus de test d'une propriété A sur un objet P.

$$\begin{tikzcd}
\begin{array}{c}\text{propriété } A \\ \text{objet } P\end{array} \arrow[rr, dashed, "T_k"] \arrow[d, "\text{représentation}"'] & & \\
\begin{array}{c}H_0\text{-contexte initial} \\ \mathcal{C}_0 \\ \langle L_0, S, (\mathcal{F}_0), A\rangle\end{array} \arrow[rr, "\text{construction}"] & & \begin{array}{c}\text{jeu de tests acceptable} \\ \mathcal{T} = \langle H, (T_n)_{n\in\mathbb{N}^*}\rangle \\ \text{sur le contexte} \\ \mathcal{C}_1 \\ \langle L_1, S, (\mathcal{F}_1), A\rangle\end{array} \arrow[u, "\text{application}"']
\end{tikzcd}$$

avec $H = H_0 \sqcup H_1$

$\mathcal{C}_1$ restriction conservative de $\mathcal{C}$.

Nous appellerons par la suite H_0 la théorie des hypothèses initiales, et H_1 celle des hypothèses de construction. Elles seront étudiées en détail au chapitre 5, ainsi que la notion de restriction (conservative) de contexte.

2.4. Quelques exemples

Nous terminons ce chapitre en présentant quelques exemples et propriétés simples concernant les jeux de tests. Ils permettront, nous l'espérons, au lecteur de se familiariser avec nos notations, et de mesurer la généralité de nos définitions. Ils illustrent le rôle, sans doute quelque peu subtil, des hypothèses H. Le début du chapitre ayant été guidé par l'examen d'un problème concret, ces exemples seront volontairement abstraits, sans aucune référence à la notion de programme.

2.4.1 Des exemples faciles...

Considérons le contexte $\mathcal{C} = \langle L, S, (\mathcal{S}), A \rangle$, avec $L = \{P\}$, $S = \mathbb{N}$, $A = \{\forall x\, P(x)\}$.

L est donc le langage du premier ordre contenant pour seul symbole non-logique le symbole de prédicat unaire P. A est la L(S)-théorie contenant pour seul axiome non-logique $\forall x\, P(x)$.

Prenons pour $(\mathcal{S})$ la famille des L(S)-structures telles que P soit d'interprétation calculable.

$\mathcal{C}$ est bien un contexte de test.

Considérons $\mathcal{C}_1 = \langle \emptyset, (\emptyset) \rangle$; $H^1 = \emptyset$ et $T_n^1 = \emptyset$ $\forall n \in \mathbb{N}$. C'est bien un jeu de tests sur $\mathcal{C}$. T_n^1

est bien une $\mathcal{E}$-expérience, et $H^1 \cup T^1_{n+1} \vdash T^1_n$, puisque $\emptyset \vdash \emptyset$. D'autre part, toute structure $\mathcal{S}$ de $(\mathcal{S})$ valide H^1, puisque $\mathcal{S} \models \emptyset$. Ce jeu de tests est peu intéressant, et sera dit vide.

Considérons $\mathcal{E}_2 = \langle H^2, (T^2_n)_{n \in \mathbb{N}} \rangle$, avec $H = \emptyset$ et $T^2_n = \{ P(0) \wedge \ldots \wedge P(n) \}$. Vérifions que c'est bien un jeu de tests sur $\mathcal{E}$. T^2_n est bien un $\mathcal{E}$-test. $H^2 \cup T^2_{n+1}$ se réécrit en $P(0) \wedge \ldots \wedge P(n+1)$ qui démontre bien $P(0) \wedge \ldots \wedge P(n)$ (cf. annexe 3) Enfin, $\mathcal{S} \models H^2$ pour tout $\mathcal{S}$ de $(\mathcal{S})$. Ce jeu de tests est intéressant au plus haut point ; en effet, puisque $S = \mathbb{N}$, pour toute structure $\mathcal{S}$ de $(\mathcal{S})$, si $\mathcal{S} \models T_n$ pour tout n, alors $\mathcal{S} \models A$! (validité asymptotique logique, cf. 3.2.)

Soit $\mathcal{E}_3 = \langle H^3, (T^3_n)_{n \in \mathbb{N}} \rangle$, $H^3 = \emptyset$, $T^3_n = \{ P(n) \wedge \ldots \wedge P(0) \}$. Il se comporte comme le précédent, car $T^3_n \dashv T^2_n$ $\forall n \in \mathbb{N}$ (équivalence absolue formelle, cf. chapitre 4)

Soit $\mathcal{E}_4 = \langle H^4, (T^4_n)_{n \in \mathbb{N}} \rangle$, $H^4 = \emptyset$,
$T^4_n = \{ P(0) \wedge P(2) \wedge \ldots \wedge P(2n),$
$P(1) \wedge P(3) \wedge \ldots \wedge P(\varphi(n)) \}$ avec $\varphi(n) = E[\frac{n}{10}]$
Il se comporte comme $\mathcal{E}^2$ (équivalence asymptotique formelle, cf. chapitre 4)

Plus généralement, soit $(a_i)_{i \in \mathbb{N}}$ une énumération (éventuellement redondante) de $\mathbb{N}$.

prenons $\mathcal{C}_5 = \langle H^5, (T_n^5)_{n\in\mathbb{N}} \rangle$, avec $H^5 = \emptyset$ et $T_n^5 = \{ I(a_0) \wedge \dots \wedge I(a_n) \}$. $\mathcal{C}^5$ se comporte comme $\mathcal{C}^2$

Considérons $\mathcal{C}_6 = \langle H^6, (T_n^6)_{n\in\mathbb{N}} \rangle$ avec $H^6 = \emptyset$ et $T_n^6 = \{ I(0), I(1), \dots, I(n) \}$. $\mathcal{C}^6$ se comporte comme $\mathcal{C}_2$, car $T_n^6 \vdash T_n^2$ $\forall n \in \mathbb{N}$.

Soit $\mathcal{C} = \langle L, S, (\mathcal{S}), A \rangle$ un contexte,

et $\mathcal{C} = \langle H, (T_n)_{n\in\mathbb{N}} \rangle$ et $\mathcal{C}' = \langle H', (T'_n)_{n\in\mathbb{N}} \rangle$ deux jeux de tests sur ce contexte. Alors

$\mathcal{C} \sqcup \mathcal{C}' = \langle H \sqcup H', (T_n \sqcup T'_n)_{n\in\mathbb{N}} \rangle$ est un jeu de tests sur $\mathcal{C}$.

En effet, si $\mathcal{S} \models H$ et $\mathcal{S} \models H'$, $\mathcal{S} \models H \sqcup H'$. D'autre part, $T_n \sqcup T'_n$ est bien un $\mathcal{C}$-test ; examinons $(H \sqcup H') \sqcup (T_{n+1} \sqcup T'_{n+1})$; cela se réécrit en $(H \sqcup T_{n+1}) \sqcup (H' \sqcup T'_{n+1})$, ce qui démontre $T_n \sqcup T'_n$.

Remarquons que le test vide $\emptyset = \langle \emptyset, (\emptyset) \rangle$ est neutre pour cette opération : $\emptyset \sqcup \mathcal{C} = \mathcal{C} \sqcup \emptyset = \mathcal{C}$

2.4.2. ... et moins faciles

Considérons maintenant le contexte $\mathcal{C} = \langle L, S, (\mathcal{S}), A \rangle$ avec $A = \{ I(0), I(1), \dots \}$, L, S et $(\mathcal{S})$ étant comme au 2.4.1.

Remarquons que, puisque $S = \mathbb{N}$,

pour toute structure $\mathcal{S}$ de $(\mathcal{S})$,

$\mathcal{S} \models \forall x\, P(x)$ si et seulement si $\mathcal{S} \models A$

Cependant (cf. annexe 3) $A \nvdash \forall x\, P(x)$!
(conséquence facile du théorème de compacité).

Examinons $\mathcal{C}_2$. Il vérifie $H \bigsqcup_{n \in \mathbb{N}} T_n \vdash A$.
En effet, fixons $k \in \mathbb{N}$. Alors $T_k \vdash P(k)$.
Donc $\bigsqcup_{n \in \mathbb{N}} T_n \vdash P(k)$, et ce pour tout k.
Donc $\bigsqcup_{n \in \mathbb{N}} T_n \vdash A$ (et $H = \emptyset$!)

Cette propriété implique bien sûr que pour toute structure $\mathcal{S}$, si $\forall n \in \mathbb{N}$, $\mathcal{S} \models T_n$, alors $\mathcal{S} \models A$. En effet, si $\mathcal{S} \models T_n$ pour tout n, $\mathcal{S}$ est un modèle de $\bigsqcup_{n \in \mathbb{N}} T_n$, et puisque A est un théorème de $\bigsqcup_{n \in \mathbb{N}} T_n$, par le théorème de validité (cf. annexe 3), $\mathcal{S} \models A$.

Remarquons qu'en 2.4.1, $\mathcal{C}_2$ ne vérifiait <u>pas</u> cette propriété, car $\forall x\, P(x)$ n'est pas démontrable à partir de $\{P(0), P(1) \dots\}$. Pour remédier à celà, il nous faut donc enrichir H, et donc restreindre $(\mathcal{S})$.

2.4.2.1 hypothèses de régularité

Sur le contexte $\mathcal{C}$ précédent, considérons le jeu de tests $\mathcal{C}^7 = \langle H^7, (T_n^7)_{n \in \mathbb{N}} \rangle$, avec
$H^7 = \{P(0) \wedge \dots \wedge P(k) \rightarrow \forall x\, P(x)\}$ et
$T_n^7 = \{P(0) \wedge \dots \wedge P(n)\}$, k étant un constante fixée.

$\mathcal{C}^7$ n'est pas un jeu de tests sur $\mathcal{C}$, car il existe des structures de $(\mathcal{S})$ ne validant pas H. Il nous faut donc considérer un contexte restreint de $\mathcal{C}$, $\mathcal{C}^1 = \langle L, S, (\mathcal{S}_1), A \rangle$ avec, par exemple, $(\mathcal{S}_1) = \{ \mathcal{S} \in (\mathcal{S}), \mathcal{S} \models H \}$ (cf. 5.1.3). Notons que cette restriction est bien conservative (cf. 2.1.2.3). En effet, si $\mathcal{S}$ est une structure de $(\mathcal{S})$ telle que $\mathcal{S} \models A$, alors $\mathcal{S} \models \{P(n)\}$ pour tout n, et donc $\mathcal{S} \models \{P(0) \wedge \ldots \wedge P(k)\}$, et donc $\mathcal{S} \models H$, soit $\mathcal{S} \in (\mathcal{S}_1)$. $\mathcal{C}^7$ vérifie alors la propriété $H \sqcup \bigsqcup_{n \in \mathbb{N}} T_n \vdash A$. En fait, il vérifie même $H \sqcup T_k \vdash A$; cette remarque est générale, comme le précise le théorème de compacité proposé en 3.2.4.

Une telle hypothèse, ou plus généralement une hypothèse de la forme :

$$\phi(a_0) \wedge \ldots \wedge \phi(a_k) \longrightarrow \forall x \, \phi(x)$$

sera appelée <u>hypothèse de régularité</u>. Le paragraphe 3.2.4. montre que, en un certain sens, cette hypothèse est minimale pour obtenir la propriété $H \sqcup \bigsqcup_{n \in \mathbb{N}} T_n \vdash A$ (validité asymptotique formelle). Remarquons que la constante k n'a aucune incidence sur le résultat. Seul compte le fait qu'<u>il existe une telle constante k</u>. Une telle constante sera dite <u>virtuelle</u>.

2.4.2.2 Hypothèses d'uniformité

Les hypothèses de régularité sont en fait un cas particulier d'hypothèses d'uniformité, applicable notamment au cas où S est "naturellement ordonné". Dans la pratique, elles s'expriment ainsi, pour RC par exemple.

"Si les assertions d'entrée et de sortie sont vérifiées pour x variant de 0 à 20, alors $\{x \geq 0\}\ RC\ \{y \geq 0,\ y^2 \leq x < (y+1)^2\}$"

Dans le cas général, suivant les heuristiques développées notamment par WEYUKER & OSTRAND [W&O 80], et utilisées plus ou moins explicitement déjà par de nombreux auteurs, nous sommes amenés à partionner S en sous-domaines d'uniformité ([Boug 82], et annexe 1). Nous restreignons le contexte en enrichissant L de nouveaux symboles de prédicat, D_k, $k \in \mathbb{N}$. $D_k(s)$ signifie que s appartient au $k^{ième}$ sous-domaine d'uniformité. Puis nous rajoutons les hypothèses suivantes

$$(\exists x\ D_k(x) \wedge P(x)) \rightarrow \forall x\ (D_k(x) \rightarrow P(x))$$

Intuitivement, si P est vrai pour une valeur de D_k, P est vrai sur tout D_k. P est donc soit vrai sur tout D_k, soit faux sur tout D_k.

Rajoutons alors à L les symboles de constantes a_k, $k \in \mathbb{N}$, et sous les hypothèses supplémentaires $\{ D_k(a_k), k \in \mathbb{N} \}$,

$T_n = \{ P(a_0) \wedge \ldots \wedge P(a_n) \}$ est un jeu de tests vérifiant, pour tout $\mathcal{S}$,

si $\forall n \in \mathbb{N}$ $\mathcal{S} \models T_n$, alors $\mathcal{S} \models A$.

Si, de plus, les D_k ne sont en fait qu'en nombre fini, nous avons :

$$H \bigsqcup_{n \in \mathbb{N}} T_n \vdash A.$$

Un tel type d'hypothèse sera dit <u>hypothèse d'uniformité</u>. Dans l'utilisation la plus courante, S tout entier est considéré comme uniforme.

$$\exists x\, P(x) \rightarrow \forall x\, P(x)$$

2.4.3. Conclusion

Par ces quelques exemples et contre-exemples, nous avons voulu permettre au lecteur de se familiariser avec nos définitions et notations avant d'aborder les chapitres plus théoriques qui suivent.

Nous avons tenu, dès à présent, à introduire les notions d'hypothèses de régularité et d'uniformité, qui jouent un rôle central

dans ce travail.

Le lecteur attentif, persévérant et critique se sera sans doute étonné que nous n'envisagions que le cas $S = \mathbb{N}$ et $A = \forall x\, P(x)$. Nous montrerons en 3.4.3. que, en un certain sens, ce sont les seuls exemples intéressants. En particulier, nous ne savons pas traiter de manière satisfaisante le cas : $A = \exists x\, P(x)$.

Plus précisément, nous ne savons traiter que le cas des formules <u>purement universelles</u>, c'est à dire telles que leur forme prénexe (cf. annexe 3) ne contienne que des quantificateurs universels.

Ce sont donc des formules équivalentes à des formules du type

$$\forall x_1 \dots \forall x_n \; \varphi(x_1 \dots x_n)$$

où φ est sans quantificateur.

Par exemple,

$$\forall x\, [\varphi(x) \to \psi(x)],$$

$$\varphi(x) \to \forall x\, \psi(x)$$

sont purement universelles, mais non

$$[\forall x\, \varphi(x)] \to \psi(x)$$

Cette restriction pourra peut-être paraître surprenante à certains lecteurs. Elle est cependant analogue à celles rencontrées dans le domaine des bases de données (cf. [Demo 82]).

2.5 Conclusion

Ce long chapitre nous a permis de présenter, pas à pas, notre modélisation de la notion de processus de test, dégagée empiriquement au chapitre premier.

Cette modélisation s'appuie sur le formalisme de la logique du premier ordre, et nous avons essayé de dégager les avantages et inconvénients de l'emploi de cet outil mathématique. D'autres, plus fins, eurent été peut-être mieux adaptés, mais il nous a semblé important d'utiliser un outil largement utilisé et diffusé, admis et reconnu par tous. D'autre part, l'emploi de cet outil nous permet d'obtenir des résultats susceptibles d'être comparés à ceux obtenus par la preuve, et donc de respecter la complémentarité du test et de la preuve formelle.

Enfin nous avons défini, de manière totalement abstraite, les notions de contexte de test et de jeu de tests. Ces notions peuvent donc maintenant être étudiées pour elles-mêmes, hors de toute référence à la notion de programme.

C'est l'objet des chapitres qui suivent.

Chapitre 3 : Propriétés fondamentales.

Le chapitre précédent a proposé une modélisation de la notion de processus de test. Ce modèle fait intervenir les notions de contexte de test et de jeu de tests, que nous avons décidé de traiter grâce à la logique du premier ordre. Nous en avons donc donné des définitions mathématiques grâce à cet outil.

Il est bien évident que, dans la pratique, le jeu de tests produit lors de l'étape de construction d'un processus devra être un "bon" jeu de tests, en ce sens qu'il devra être capable de fournir des informations, les plus précises possibles, à l'utilisateur. Il nous appartient donc maintenant, dans le cadre mathématique que nous avons tracé, et d'après la discussion du chapitre 1, de définir rigoureusement les qualités fondamentales d'un jeu de tests, afin de pouvoir, par la suite, discerner si un jeu de tests est "bon".

Les définitions que nous allons dégager sont directement inspirées de celles proposées par GOODENOUGH & GERHART, exposées en détail en annexe 1. Nous nous intéresserons aux notions de <u>fiabilité</u>, de <u>validité</u> et

de <u>biais</u>, à partir desquelles nous définirons la notion de <u>jeu de tests acceptable</u>.

Ces définitions seront données dans un cadre un peu plus général que celui d'un jeu de tests, celui d'une famille de tests.

<u>définition</u>: famille de tests.

Soit $\mathcal{C} = \langle L, S, (\mathcal{S}), A \rangle$ un contexte de test.

Une famille de tests $\mathcal{F}$ est la donnée de

- une $L(S)$-théorie H telle que toute structure $\mathcal{S}$ de $(\mathcal{S})$ soit modèle de H
- une famille de $L(S)$-théories $(T_i)_{i \in I}$

L'on ne fait donc aucune hypothèse sur les T_i, ni sur I. Une telle famille $\mathcal{F}$ sera notée $\langle H, (T_i)_{i \in I} \rangle$.

Toutes les définitions données seront donc valables sans modification pour un jeu de tests.

3.1. Notion de fiabilité.

Rappelons tout d'abord la notion classique de fiabilité, telle que WHITE la transcrit [Whit 81]

<u>définition</u> : fiabilité

Un test T est réussi par un programme P si P est correct sur tout élément de T.
Soit T_1 et T_2 deux tests satisfaisant un critère de sélection C pour un programme P. C est fiable si T_1 est réussi par P ssi T_2 l'est.

3.1.1. Discussion

Cette définition nous semble judicieuse en ce sens qu'elle associe la notion de fiabilité à un <u>critère de sélection</u>, et non à un test, ce qui rejoint précisément les conclusions du chapitre 1. Cependant, nous avons dégagé l'importance de posséder un critère non pas uniforme, plat, comme celui que définissent GOODENOUGH & GERHART, mais ordonné par la qualité des différents tests qui le composent. La définition précédente peut être vue comme une assertion sur la cohérence d'un critère, son homogénéité, son uniformité : quel que soit le test sélectionné

par C, le résultat obtenu lors de l'expérience proposée par le test est le même.

Dans le cas d'une famille de tests $\mathcal{T} = \langle H, (T_i)_{i \in I} \rangle$ sur un contexte $\mathcal{C}$, I étant muni d'un ordre partiel (ordre de qualité), la cohérence se traduira par le fait suivant. Si $i_1 > i_2$, alors T_{i_1} sera un test plus précis, plus contraignant que i_2, en ce sens que si T_{i_1} est réussi par P, alors T_{i_2} le sera. Remarquons que le fait d'ordonner la famille de test nous amène à introduire une <u>anistropie</u>, privilégiant la réussite d'un test par rapport à son échec.

Conformément à notre principe dégagé au chapitre 1, nous considérons des propriétés uniformes par rapport à la famille de structures du contexte.

3.12. Définitions.

Ces remarques nous conduisent donc aux définitions suivantes.

<u>définition</u>: fiabilité logique projective

Soit $\mathcal{C} = \langle L, S, (\mathcal{S}), A \rangle$ un contexte de test, et $\mathcal{T} = \langle H, (T_i)_{i \in I} \rangle$ une famille de tests. $\mathcal{T}$ est dite projectivement logiquement fiable si, I étant ordonné partiellement

par $>$,

pour toute structure $\mathcal{S}$ de $(\mathcal{S})$,

pour tout i_1 et i_2 de I tels que $i_1 > i_2$

si $\mathcal{S} \models T_{i_1}$ alors $\mathcal{S} \models T_{i_2}$

Par dualité, nous obtenons immédiatement la définition suivante.

<u>définition</u> : fabilité formelle projective

Soit $\mathcal{C} = \langle L, S, (\mathcal{S}), A \rangle$ un contexte de test, et $\mathcal{F} = \langle H, (T_i)_{i \in I} \rangle$ une famille de tests, I étant partiellement ordonné par $>$.

$\mathcal{F}$ est dite formellement projectivement fiable si

pour tout i_1, i_2 de I tels que $i_1 > i_2$

$H \sqcup T_{i_1} \vdash T_{i_2}$

Nous avons immédiatement la proposition suivante

<u>proposition</u> :

Soit $\mathcal{C}$ un contexte de test, $\mathcal{F}$ une famille de tests sur ce contexte.

Si $\mathcal{F}$ est formellement projectivement fiable, $\mathcal{F}$ est logiquement projectivement fiable.

La réciproque est bien évidemment fausse en général (cf annexe 3 : non-catégoricité de la logique du premier ordre)

Nous avons choisi le qualificatif "projectif" par analogie avec la notion de limite projective. Un vocabulaire plus adapté pourrait sans aucun doute être proposé.

3.1.3 Jeu de tests et fabilité

Par définition, nous avons imposé à un jeu de tests d'être indexé par un ensemble dénombrable, $\mathbb{N}$ en fait, et d'être formellement projectivement fiable pour l'ordre (total) de $\mathbb{N}$ (cf. 2.3.5.)

Il ne fait pas de doute que, dans la pratique, un jeu de tests, pour être rationnellement utilisable, doit être projectivement fiable; en d'autres termes, plus l'on consacre d'argent et de temps à tester un programme, plus on est assuré de la correction de celui-ci. Cependant, nous aurions pu admettre des jeux de tests seulement logiquement projectivement fiables, ou bien indexés de manière plus complexe. Notre choix est le plus simple sur le plan théorique, mais trop restrictif peut-être pour la pratique. Cependant, les exemples proposés au chapitres 6 et 7 montrent qu'il conduit à des résultats pratiquement exploitables.

3.2. Notion de validité.

Rappelons tout d'abord la définition proposée en [Whit 87].

<u>définition</u> : validité

Soient P et C comme au 3.1. Supposons P incorrect. C est dit valide s'il existe un test T sélectionné par C ne réussissant pas sur P.

3.2.1 Discussion

Comme pour la définition de la fabilité, il nous semble fondamental, suivant en cela GOODENOUGH & GERHART, de définir la notion de validité pour un <u>critère</u>, et non pour un test.

Cependant, cette définition ne prend pas du tout en compte la notion de <u>qualité</u> que nous avons introduite au chapitre 1. La définition précédente impose au critère de sélectionner un test de qualité infinie, idéale. Or, dans la pratique, si par hasard un tel test existe, il est de coût largement prohibitif, et donc virtuellement inexistant (cf 1.4.2.).

L'important n'est donc pas l'existence d'un test idéal, mais l'existence de tests

de qualité aussi grande que désiré (et probablement de prix en conséquence). A l'utilisateur, dans la phase d'application, de choisir, selon ses propres exigences de coût, un test de rentabilité optimale dans la famille.

De manière analogue, en mathématique, il est souvent plus rentable d'être capable d'approcher un objet avec une précision arbitraire ("epsilonnesque") que de le définir parfaitement.

Nous sommes donc conduit à une notion asymptotique de la validité, assurant que, <u>à l'infini éventuellement</u>, il existe un test mettant en évidence l'incorrection de $\mathcal{P}$.

3.2.2. Définitions

<u>définition</u> : validités asymptotiques logique et formelle

Soit $\mathcal{E} = \langle L, S, (\mathcal{S}), A \rangle$ un contexte de test, et $\mathcal{F} = \langle H, (T_i)_{i \in I} \rangle$ une famille de tests.

$\mathcal{F}$ est logiquement asymptotiquement valide

si

pour toute structure $\mathcal{S}$ de $(\mathcal{S})$,

si $\mathcal{S} \models T_i$ pour tout $i \in I$, alors $\mathcal{S} \models A$

$\mathcal{F}$ est formellement asymptotiquement valide

si $\bigsqcup_{i \in I} T_i \sqcup H \vdash A$

La seconde partie est bien sûr la définition duale de la précédente. Nous avons donc la propriété classique suivante.

<u>proposition</u>:

Soit $\mathcal{C}$ un contexte de test, et $\mathcal{F}$ une famille de tests sur ce contexte.

Si $\mathcal{F}$ est formellement asymptotiquement valide, alors $\mathcal{F}$ est logiquement asymptotiquement valide

Si $\mathcal{F}$ est en fait un jeu de tests, alors $I = \mathbb{N}$, et la famille $\langle H, (T_n)_{n \in \mathbb{N}} \rangle$ est formellement projectivement fiable. Les définitions se réécrivent alors ainsi.

Pour tout $\mathcal{S} \in (\mathcal{S})$, il existe un $n_{\mathcal{S}}$ tel que

$$(\forall n \geq n_{\mathcal{S}} \quad \mathcal{S} \models T_n) \Rightarrow \mathcal{S} \models A$$

et

$$\exists n_0 \quad \bigsqcup_{n \geq n_0} T_n \sqcup H \vdash A .$$

Ces formes dérivées montrent de manière plus claires sans doute que le test parfait est potentiellement rejeté à l'infini.

La réciproque de la proposition est fausse en général comme le montre l'exemple important suivant. Considérons $S = \mathbb{N}$, $H = \phi$, $A = \{\forall x P(x)\}$ (avec les abus de langages justifiés

par ailleurs), et $(\mathcal{S})$ la famille des $\{\mathcal{R}\}$-structures sur $\mathbb{N}$ telles que $\mathcal{I}$ soit calculable. Alors $T_n = \{\mathcal{I}(0) \wedge \ldots \wedge \mathcal{I}(n)\}$ est asymptotiquement logiquement valide.

En effet, considérons $\mathcal{S} \in (\mathcal{S})$. Soit n fixé. Si $\mathcal{S} \models T_n$, alors $\mathcal{S} \models \mathcal{I}(n)$ et donc $\mathcal{R}_{\mathcal{S}}(n)$, et ceci pour tout n. Donc $\mathcal{S} \models \forall x\, \mathcal{I}(x)$ et $\mathcal{S} \models A$. Mais nous avons montré en annexe 3 que $\{\mathcal{I}(i), i \in \mathbb{N}\}$ <u>ne démontre pas</u> $\{\forall x\, \mathcal{I}(x)\}$ (c'est une application facile du théorème de compacité de la logique égalitaire du premier ordre).

3.2.3 Validité finie

Un cas particulier important des définitions précédentes est celui où le test parfait est en fait à distance finie, comme le proposaient GOODENOUGH & GERHART. Cette notion de "distance" n'a bien sûr de sens que dans le cas d'une famille projectivement finie. Nous pouvons cependant, par souci de généralité, poser les définitions suivantes.

<u>definitions</u> : validités finies logique et formelle

Soient $\mathcal{E} = \langle L, S, (\mathcal{S}), A \rangle$ un contexte de test, et $\mathcal{F} = \langle H, (T_i)_{i \in I} \rangle$ une famille de tests.

$\mathcal{F}$ est dite finiment logiquement valide

si, pour toute structure $\mathcal{S}$ de $(\mathcal{S})$,
il existe $i_{\mathcal{S}} \in I$ tel que
si $\mathcal{S} \models i_{\mathcal{S}}$ alors $\mathcal{S} \models A$

$\mathcal{F}$ est dite formellement finiment valide
si il existe I_0 sous-ensemble fini de I
tel que

$$\bigsqcup_{i \in I_0} T_i \sqcup H \vdash A$$

<u>proposition</u> :
La validité finie formelle (resp. logique) entraîne la validité asymptotique formelle (resp. logique).
La validité finie formelle entraîne la validité finie logique, si I est un treillis, et si $\mathcal{F}$ est projectivement fable formellement.

En effet, soit alors i_0 un majorant de I_0 (sous-ensemble fini de I !). Alors

$$T_{i_0} \sqcup H \vdash T_i \qquad \forall i \in I_0.$$

donc $T_{i_0} \sqcup H \vdash \bigsqcup_{i \in I_0} T_i$ et

$$T_{i_0} \sqcup H \vdash H.$$

Donc $T_{i_0} \sqcup H \vdash \bigsqcup_{i \in I_0} T_i \sqcup H \vdash A$, et l'on peut prendre $i_{\mathcal{S}} = i_0$ pour tout $\mathcal{S}$.

Dans le cas d'un jeu de tests, la validité finie formelle s'exprime par le fait suivant.

$\exists n_0 \quad T_{n_0} \sqcup H \vdash A$.

Alors, par le raisonnement précédent, pour tout $n \geq n_0$, $T_n \sqcup H \vdash A$.

Si la famille $\mathcal{F} = \langle H, (T_i)_{i \in I} \rangle$ est finiment logiquement valide, alors le critère $C = \{T_i, i \in I\}$ est valide au sens de [GGG 75]

En effet, si $\mathcal{P}$ ne vérifie pas A, alors la structure $\mathcal{S}$ représentant abstraitement $\mathcal{P}$ ne valide pas A. Il suffit de prendre $T_{i_\mathcal{S}}$ pour test dans ce cas.

3.2.4 Théorème de compacité

Le théorème suivant montre que, dans les cas simples, tout jeu de tests formellement asymptotiquement valide est en fait formellement finiment valide, et donc valide au sens de GOODENOUGH & GERHART

<u>Théorème de compacité</u>

Soit $\mathcal{C}$ un contexte de test, et $\mathcal{F}$ une famille de tests sur $\mathcal{C}$.

Soit $\mathcal{C} = \langle L, S, (\mathcal{S}), A \rangle$, et supposons que A ne contienne qu'un <u>nombre fini d'axiomes non-logiques</u>

Alors, si $\mathcal{F}$ est formellement asymptotique-

ment valide, σ est en fait finiment formellement valide.

<u>démonstration</u>

C'est une application directe du théorème de compacité de la logique (égalitaire) du premier ordre (cf. annexe 3, paragraphe 8.6.)

Nous avons $H \sqcup_{i \in I} T_i \vdash A$.

Soient $a_1 \dots a_n$ les axiomes non-logiques de A. $H \sqcup_{i \in I} T_i \vdash a_k$ pour k fixé. Donc a_k est en fait conséquence formelle d'un nombre fini d'axiomes non-logiques de $H \sqcup_{i \in I} T_i$, et donc de $H \sqcup_{i \in I_k} T_i$, avec I_k fini.

Donc, en posant $I_0 = I_1 \sqcup \dots \sqcup I_n$, qui est bien fini, nous obtenons

$$H \sqcup_{i \in I_0} T_i \vdash \{a_1 \dots a_n\}, \text{ et donc}$$
$$H \sqcup_{i \in I_0} T_i \vdash A.$$

▨

Ce résultat justifie l'exemple du 3.2.2. Cet exemple montre de plus que ce théorème n'admet pas de dual logique. Un jeu de tests peut très bien être asymptotiquement valide logiquement, sans être finiment logiquement valide, A ne contiendrait-il qu'un seul axiome. Cela justifie donc, a posteriori, le choix que nous avons fait, dans la définition d'un jeu de tests, de <u>privilégier l'aspect formel</u> par rapport à

l'aspect logique.

Ce théorème nous donne par ailleurs un indication sur la forme des jeux de tests asymptotiquement formellement valide dans le cas où A n'a qu'un nombre fini d'axiomes non-logique. Par fiabilité projective, on a alors, pour un certain n_0, $H \sqcup T_{n_0} \vdash A$, où T_{n_0} est (f)-effectivement vérifiable. Intuitivement, cela signifie que A se factorise en une partie "calculable", T_{n_0}, et une partie "transcendante", H. Nous retrouvons donc la discussion du chapitre 1. En identifiant T_{n_0} et A à la conjonction de leurs axiomes non logiques (en nombre fini !), le théorème de déduction montre que

$$H \vdash T_{n_0} \rightarrow A. \qquad \text{(cf annexes, paragraphe 8.2.)}$$

Intuitivement, H contient donc <u>au moins</u> une hypothèse d'uniformité, de la forme

"partie calculable" $\rightarrow$ "propriété sous test"

permettant de généraliser l'expérience.

Dans le cas de l'exemple cité en 3.2.2, $T_{n_0} = P(0) \wedge \ldots \wedge P(n_0)$, et $A = \forall x P(x)$. Cette discussion montre que l'hypothèse

$$H = \{P(0) \wedge \ldots \wedge P(n_0) \rightarrow \forall x P(x)\}$$

est <u>minimale</u>, en un certain sens, pour que $\langle H, (T_n)_{n \in \mathbb{N}} \rangle$ soit asymptotiquement formellement valide. Remarquons cependant que, paradoxa-

lement, peut-être, toute hypothèse de la forme

$H = \{P(0) \wedge \dots \wedge P(k) \dashrightarrow \forall x\, P(x)\}$

fait de $\langle H, (T_n)_n \rangle$ un jeu de tests asymptotiquement formellement valide, indépendamment de la valeur de k. Une telle hypothèse sera dite hypothèse de régularité. k sera appelé une <u>constante virtuelle</u>.

3.2.5 Conclusion

A partir de la définition habituelle de la validité d'un critère de sélection, nous avons proposé plusieurs notions, plus fines : validité asymptotique, logique et formelle, validité finie, logique et formelle.

La validité finie est un cas particulier de la validité asymptotique, et rejoint la définition classique de la validité.

Le théorème de compacité montre que, dans les cas simples, la validité asymptotique formelle se réduit à la validité classique. Ce même théorème montre que, dans ces mêmes cas, si un jeu de tests $\langle H, (T_n)_{n \in \mathbb{N}} \rangle$ est valide (asymptotiquement formellement), alors H démontre <u>nécessairement</u> une hypothèse d'uniformité de la forme "partie calculable" $\longrightarrow$ "propriété sous test". De telles hypothèses ont déjà été étudiées au chapitre 2 (cf. 2.4.2.)

3.3 Notion de biais

Nous développons dans ce paragraphe une notion extrêmement importante qui semble, en première analyse, avoir échappé au théoricien du test cité par [Whit 81] : la notion de biais

Cette notion a été signalée plusieurs fois au chapitre 1, et joue un rôle fondamental en statistique (cf. 1.5.).

Rappelons la définition donnée au chapitre 1.

<u>définition</u> :

Un test δ est dit sans biais si

$$\inf_{\theta \in \Theta_1} \Pi_\delta(\theta) \geq \sup_{\theta \in \Theta_0} \Pi_\delta(\theta) = \alpha_\delta$$

avec $\Pi_\delta(\theta) = E_\theta(\delta)$

En termes plus clairs pour notre lecteur, peut-être, si l'évènement a été produit par un $\theta \in \Theta_1$, le test a uniformément plus de chances de choisir H_1 plutôt que H_0 ; si un $\theta \in \Theta_0$ était en cause, il a uniformément plus de chances de choisir H_0 plutôt que H_1

Dans notre formalisme, où nous n'avons pas, à ce niveau, introduit la notion du "probable", nous dirons simplement qu'un

test est biaisé s'il existe une structure, vérifiant l'hypothèse, qui ne soit pas modèle d'un des tests ; en d'autres termes, si l'application du jeu de tests au problème nous amène (potentiellement) à conclure que $\mathcal{P}$ ne vérifie pas A alors qu'en réalité $\mathcal{P}$ vérifie A. Remarquons dès à présent l'<u>anisotropie</u> de la définition qui ne dit rien du cas où $\mathcal{P}$ ne vérifie réellement pas A.

3.3.1. Définitions

Nous sommes donc conduit naturellement aux définitions suivantes.

<u>définitions</u> : jeu de tests formellement et logiquement non biaisés

Soit $\mathcal{C} = \langle L, S, (\mathcal{S}), A \rangle$ un contexte de test.
et $\mathcal{T} = \langle H, (T_i)_{i \in I} \rangle$ une famille sur ce contexte.

$\mathcal{T}$ est logiquement non biaisé si
pour toute structure $\mathcal{S}$ de $(\mathcal{S})$
si $\mathcal{S} \models A$, alors $\mathcal{S} \models T_i \quad \forall i \in I$.

$\mathcal{T}$ est formellement non biaisé si
$H \sqcup A \vdash T_i \quad \forall i \in I$

La seconde définition est duale de la première, et nous pouvons énoncer la propriété suivante.

<u>proposition</u> :

L'absence de biais formelle implique l'absence de biais logique.

En fait, ces deux définitions sont les <u>réciproques</u> de celles de la validité asymptotique (cf. 3.2.2.).

Si les travaux cités en [Whit 81] ne prennent pas en compte la notion de biais, c'est en fait parce qu'elle est implicite dans les tests qu'ils considèrent. Pour eux un test est un sous-ensemble T fini du domaine D. L'expérience <u>implicite</u> associée est $\forall t \in T \; OK(t)$: le programme est correct sur toutes les valeurs de T. La propriété implicite à tester est $\forall d \in D, \; OK(d)$. Comme $\forall d \in D, OK(d) \rightarrow \forall t \in T \; OK(t)$ leur notion de test, traduite dans notre formalisme, ne s'intéresse qu'à des jeux de tests non-biaisés ! Notre approche, beaucoup plus générale et abstraite que la leur, nous permet donc de mettre en lumière des propriétés restées implicites jusqu'alors.

3.3.2. Exemples.

Considérons maintenant quelques exemples.

Prenons $S = \mathbb{N}$, $H = \emptyset$, $A = \{\forall x\,(x=0 \vee P(x))\}$, $(\mathcal{S})$ la famille des $\{P\}$-structures sur $\mathbb{N}$ telles que P soit d'interprétation calculable.

Alors $T_n = \{P(0) \wedge \ldots \wedge P(n)\}$ est un jeu de tests logiquement asymptotiquement valide, mais biaisé, car l'on peut choisir une structure $\mathcal{S}_0$ de $(\mathcal{S})$, telle que $P(n)$ pour $n \neq 0$, et $\neg P(0)$ (prendre $P_{\mathcal{S}}(x) = \neg x = 0$). $\mathcal{S}_0 \models A$, mais $\mathcal{S}_0 \not\models T_n$ quel que soit $n \in \mathbb{N}$.

Par contre $T_n = \{P(1) \wedge \ldots \wedge P(n)\}$ pour $n \geq 1$, et $T_0 = \emptyset$ est logiquement asymptotiquement valide et formellement non biaisé.

Les jeux de tests de propriétés temporelles sont, en général, biaisés. Un cas typique est le suivant. Prenons $S = \mathbb{N}$, $H = \{\forall x\,(Q(x) \rightarrow P(x))\}$ $A = \{P(x_0)\}$, $(\mathcal{S})$ étant la famille des structures telles que Q soit d'interprétation calculable, et non P. Alors $T_n = \{Q(x_0)\}$ est un jeu de tests formellement asymptotiquement valide, mais biaisé en général. Il est logiquement non biaisé (mais formellement biaisé) si les structures de $(\mathcal{S})$ vérifient en fait $\forall x\,(Q(x) \leftrightarrow P(x))$.

3.4. Notion d'acceptabilité

Nous sommes maintenant en mesure de définir ce qu'est un "bon" test. Examinons tout d'abord la définition qui en est proposée par GOODENOUGH & GERHART [G&G 75].

<u>définition</u> : idéalité

Un critère est idéal par rapport à P, si il est fiable et valide.

Cette définition était justifiée par le théorème suivant ("théorème fondamental du test").

<u>théorème</u> :

Soit P un programme, et C un critère idéal pour P. Si P est incorrect, alors tout test T sélectionné par C le mettra en évidence.

3.4.1. Discussion

Plusieurs remarques peuvent être faites à propos de cette définition.

Tout d'abord, elle ne définit pas ce qu'est un "bon" test, mais ce qu'est un "bon" <u>critère de test pour un programme donné</u>. C'est là, pour notre part, un point décisif, tout à fait

conforme aux conclusions du chapitre 1.

Cependant, elle ne prend pas en compte l'un des problèmes cruciaux du test en informatique, qui est l'existence d'un oracle de décision, permettant de conclure à la réussite ou à l'échec du test en un temps fini [Weyu 80]. Il est alors extrêmement facile de définir un critère c idéal en posant que T est sélectionné par c si T contient au moins une valeur traitée incorrectement par P. Mais ce critère sera sans doute peu utilisable pratiquement.

Cette définition est par ailleurs trop brutale, en ce sens qu'elle réduit la notion de test à celle de preuve. En fait, elle est bien adaptée à l'utilisation mathématique de la notion de test, étudiée en 1.1. . En effet, exhiber un critère idéal revient à prouver, <u>selon un schéma de preuve très particulier</u>, la correction du programme, ce qui est souvent irréalisable effectivement. Ainsi, la notion de test est réduite à n'être qu'un pâle substitut de celle de preuve. Or, le problème de la preuve formelle est précisément l'absence d'intermédiaire entre le "correct" et l'"incorrect".

A nous, donc, de reprendre à notre compte la notion de "presque sûrement correct", d'"asympto-

tiquement correct", de " correct à l'infini".
Un bon critère doit non pas démontrer la correction, mais l'approcher aussi finement que l'utilisateur le désire.

3.4.2. Définitions

A partir de cette étude critique de la notion d'idéalité, nous pouvons déterminer les caractéristiques d'un "bon" jeu de tests.

Pour nous, un jeu de tests est, par définition, formellement projectivement fiable, et effectivement calculable sur la famille de structures considérées, ce qui répond au problème de l'oracle. Pour rejoindre la notion précédente de test, il faut imposer l'absence de biais, alors implicite. Enfin, nous imposons la propriété déduite de la notion de validité, la validité asymptotique, d'où les définitions suivantes.

<u>définition</u> : acceptabilité logique et formelle.

Un jeu de tests $\mathcal{T}$ est logiquement acceptable s'il est

- logiquement asymptotiquement valide
- logiquement non biaisé.

Un jeu de tests $\mathcal{T}$ est formellement

acceptable s'il est
- formellement asymptotiquement valide
- formellement non biaisé.

<u>proposition</u> :
L'acceptabilité formelle entraine l'acceptabilité logique.

Avec les notations habituelles, un jeu de tests logiquement acceptable vérifie donc pour toute structure $\mathcal{S}$ de $(\mathcal{S})$

$$\mathcal{S} \models A \text{ si et seulement si } \mathcal{S} \models T_n \quad \forall n \in \mathbb{N}$$

De même, un jeu de tests formellement acceptable vérifie :

$$\begin{cases} H \sqcup A \vdash T_n & \forall n \in \mathbb{N} \\ H \sqcup \bigsqcup_{n \in \mathbb{N}} T_n \vdash A & , \end{cases}$$

soit $\quad H \bigsqcup_{n \in \mathbb{N}} T_n \vdash H \sqcup A$

En d'autres termes, A et $\bigsqcup_{n \in \mathbb{N}} T_n$ sont H-formellement équivalents (cf. annexe 3).

Pour nous, un "bon" jeu de tests sera un jeu de tests <u>formellement acceptable</u> (nous dirons simplement acceptable). Ce choix est cohérent avec notre démarche qui a privilégié les aspects formels et théoriques par rapport aux aspects logiques et pratiques (cf. chapitre 2, et 3.2.4).

3.4.3 Exemples

Examinons quelques exemples typiques. Ils montrent que, en un certain sens, les propriétés admettant "naturellement" un jeu de tests acceptable sont rares ; typiquement, ce sont des formules purement universelles dont la matrice est d'interprétation calculable sur toutes les structures envisagées, l'ensemble des individus étant dénombrables.

3.4.3.1 Propriétés temporelles.

Elles correspondent au cas où $A = \{I(x_0)\}$, I étant d'interprétation non calculable. Nous avons vu, en 3.3.2., que les jeux de tests "naturels" pour de telles propriétés sont en général formellement biaisés, et donc ne sont pas acceptables.

3.4.3.2 Propriétés spatiales universelles pures

C'est le cas où $A = \{\forall x_1 \ldots \forall x_n \ \phi(x_1 \ldots x_n)\}$ ϕ étant composée de symboles d'interprétation calculable sur chacune des structures considérées.

Examinons le cas $n = 1$.

Prenons $S = \mathbb{N}$, $H = \phi$ et $A = \{\forall x \ I(x)\}$

Alors le jeu de tests "naturel" :

$T_n = \{ P(0) \wedge \dots \wedge P(n) \}$ n'est pas formellement asymptotiquement valide, et donc n'est pas acceptable.

En 3.2.4., nous avons montré qu'il faut, au moins, introduire une hypothèse d'uniformité pour le rendre acceptable. Cette hypothèse, dans ce cas ($\mathbb{N}$ possédant un ordre naturel), sera $H = \{ P(0) \wedge \dots \wedge P(k) \rightarrow \forall x\, P(x) \}$

Le jeu de tests est alors acceptable.

De manière duale, l'on peut aussi affaiblir A pour conserver $H = \emptyset$. Nous prendrons alors $A = \{ P(i),\ i \in \mathbb{N} \}$, qui est ($\mathcal{S}$)-logiquement équivalent au précédent, mais non formellement équivalent (cf. annexe 3).

$T_n = \{ P(0) \wedge \dots \wedge P(n) \}$ est alors acceptable.

Dans le cas non dénombrable, il n'existe pas de jeu de tests "naturel" de $\forall x\, P(x)$. On prendra en général une hypothèse d'uniformité de la forme :

$$\exists x [P(x) \wedge D(x)] \rightarrow \forall x\, P(x),$$

où D est un prédicat assurant que x est <u>représentatif</u> du comportement de P. Un jeu de tests acceptable sera donné par

$T_n = \{ P(a) \}$, avec a tel que $D(a)$.

3.4.3.3 propriétés spatiales existentielles.

Dans ce paragraphe, nous montrons que, intuitivement, nous ne savons pas tester les propriétés existentielles.

Considérons $S = \mathbb{N}$, $L = \{P\}$, $A = \{\exists x\, P(x)\}$, et considérons la famille $(\mathcal{S})$ des $L(S)$-structures telles que P soit d'interprétation calculable. Soit (T_n) un jeu de tests logiquement non biaisé. Alors, pour tout $\mathcal{S}$, si $\mathcal{S} \models A$, $\mathcal{S} \models T_n$ $\forall n \in \mathbb{N}$.

En mettant les axiomes non-logiques des T_n sous forme prénexe, et en séparant les conjonctions (cf. annexes), en réduisant les sous-formules de la forme $k = p$ et $\neg k = p$ à T ou F suivant les cas, nous pouvons supposer que les axiomes non-logiques de T_n sont de la forme suivante, toutes les constantes étant distinctes : $P(n_1) \vee \ldots \vee P(n_p) \vee \neg P(m_1) \vee \ldots \vee \neg P(m_q)$.

Considérons alors l'une de ces formes, et prenons $\mathcal{S}$ telle que $P_{\mathcal{S}} = (x = m_1) \vee \ldots \vee (x = m_q)$.

$\mathcal{S} \models A$, et $\mathcal{S} \not\models T_n$. Donc les axiomes sont de la forme :

$$P(n_1) \vee \ldots \vee P(n_p)$$

Prenons alors $P_{\mathcal{S}} = (x = m)$ avec $m \neq n_1 \ldots m \neq n_p$. T_n se réduit donc à $\emptyset$, et

le seul jeu de tests logiquement

(et donc formellement) non biaisé est le jeu de tests vide.

Par contre, la "plupart" des jeux de tests sont formellement asymptotiquement valides. Par exemple $H = \emptyset$ et $T_n = \{P(0)\}$ $\forall n \in \mathbb{N}$

Pour obtenir un jeu de tests acceptable, il nous faut poser une hypothèse d'uniformité, et donc restreindre la famille $(\mathcal{F})$. On pourra par exemple prendre :

$$\exists x\, P(x) \rightarrow P(0) \vee \ldots \vee P(k).$$

Le jeu de tests $T_n = \{P(0) \vee \ldots \vee P(k)\}$ est alors acceptable.

Notons que la contraposée de l'hypothèse s'écrit : $\neg P(0) \wedge \ldots \wedge \neg P(k) \rightarrow \forall x\, \neg P(x)$. Fondamentalement, c'est donc une hypothèse de régularité.

L'on pourra s'étonner de cette différence entre le cas universel et le cas existentiel. Elle est en fait due à ce que le formalisme de la logique du premier ordre introduit cette dissymétrie en posant qu'une structure valide une théorie si elle valide chacune de ses formules, et non pas l'une d'entre elles.

Notons enfin une différence fondamentale entre le cas universel et le cas existentiel.

Dans le cas universel, les hypothèses de régularité et d'uniformité sont conservatives (cf. 2.1.2.3) En effet,

$$\forall x\, P(x) \vdash [P(0) \wedge \dots \wedge P(k) \rightarrow \forall x\, P(x)]$$
$$\forall x\, P(x) \vdash [\exists x\, P(x) \rightarrow \forall x\, P(x)].$$

Par contre, dans le cas existentiel, les hypothèses "naturelles" <u>ne sont pas conservatives</u>. En reprenant l'hypothèse citée plus haut, la structure $\mathcal{S}$ telle que $P_{\mathcal{S}} = (x = k+1)$ vérifie $\exists x\, P(x)$, mais pas

$$\exists x\, P(x) \rightarrow P(0) \vee \dots \vee P(k).$$

Lors de la construction d'un jeu de tests, nous ne serons donc pas habilité à restreindre le contexte par une telle hypothèse, contrairement au cas universel. Nous ne pourrons donc pas, en général, construire de jeu de tests acceptable lors du processus de test d'une propriété existentielle.

3.4.3.4. test par mutation

Cette méthode de test de programme a été proposée par BUDD, DeMILLO, LIPTON et SAYWARD, et a donné lieu à des réalisations fort intéressantes. Le lecteur en trouvera une description en [Boug 82], ainsi qu'en [Budd 81].

Cette méthode peut se résumer ainsi, dans notre formalisme.

Soit $\mathcal{C} = \langle L, S, (\mathcal{S}), A \rangle$ un contexte de test, et $\mathcal{T} = \langle H, (T_n)_{n \in \mathbb{N}} \rangle$ un jeu de tests sur ce contexte.

Supposons que, dans $(\mathcal{S})$, il n'existe qu'<u>une seule structure</u> $\mathcal{S}_0$ telle que $\mathcal{S}_0 \models A$

Supposons que la famille $(T_n)_{n \in \mathbb{N}}$ soit discriminante en $\mathcal{S}_0$, c'est à dire que pour tout $\mathcal{S}$ de $(\mathcal{S})$, différent de $\mathcal{S}_0$, il existe $n_{\mathcal{S}}$ tel que si $\mathcal{S} \models T_{n_{\mathcal{S}}}$, alors $\mathcal{S}_0 \not\models T_{n_{\mathcal{S}}}$ ou tel que si $\mathcal{S} \not\models T_{n_{\mathcal{S}}}$, alors $\mathcal{S}_0 \models T_{n_{\mathcal{S}}}$.

Supposons que $\mathcal{C}$ soit logiquement non biaisé.

Alors $\mathcal{C}$ est logiquement asymptotiquement valide (et donc logiquement acceptable). En effet, puisque $\mathcal{C}$ est logiquement non biaisé, $\mathcal{S}_0 \models T_n$ $\forall n \in \mathbb{N}$. Donc, pour tout $\mathcal{S} \neq \mathcal{S}_0$, il existe $n_{\mathcal{S}}$ tel que $\mathcal{S} \not\models T_{n_{\mathcal{S}}}$.

Donc, si $\mathcal{S} \models T_n$ $\forall n \in \mathbb{N}$, alors $\mathcal{S} = \mathcal{S}_0$, et $\mathcal{S} \models A$ par hypothèse.

3.4.3.5 Conclusion

Ces exemples montrent les différentes conséquences de notre définition. Nous retiendrons particulièrement que les seules propriétés susceptibles d'être "bien" testées sont les propriétés spatiales purement universelles. Les propriétés existentielles, particulièrement, ne peuvent être testées que par ajout d'hypothèses très restrictives, et surtout non conservatives (cf. chapitre 5), en ce sens qu'elles écartent des structures potentiellement vérifiant la propriété sous test.

Il est remarquable que la forme sus-citée soit précisément celle des axiomes d'un type abstrait algébrique. Le chapitre 7 montrera, en effet, que notre formalisme est particulièrement bien adapté à l'étude des spécifications algébriques abstraites.

3.4.4 Acceptabilité et couplage

Maintenant que nous avons défini ce que nous entendons par "bon" jeu de tests, nous pouvons reprendre l'étude de la notion de processus de test, exposée en 2.1.4.

L'acceptabilité formelle entraînant l'acceptabilité logique, tous les résultats exposés précédemment restent valables.

Remarquons seulement que la validité du principe se trouve renforcée par la considération des propriétés formelles plutôt que logiques. En effet, l'acceptabilité d'un jeu de tests $\mathcal{T} = \langle H, (T_n)_{n \in \mathbb{N}} \rangle$ sur un contexte $\mathcal{C} = \langle L, S, (\mathcal{S}), A \rangle$ ne dépend que de A, H et $(T_n)_{n \in \mathbb{N}}$, et absolument pas de la manière dont le jeu de tests est utilisé (c'est-à-dire $(\mathcal{S})$). La rigueur du raisonnement, est donc <u>complètement disjointe</u> des données auxquelles il est appliqué.

Cette remarque justifie a posteriori la préférence que nous avons donné à l'acceptabilité formelle par rapport à son dual logique.

3.4.5 Conclusion

Cette partie nous a permis de définir rigoureusement ce que nous entendons par la notion de "bon" jeu de tests.

Quoiqu'inspirée par la notion d'idéalité, notre définition s'en écarte sensiblement pour prendre en compte la spécificité de la notion de test : le "probable", alors que la notion d'idéalité ne proposait aucun moyen terme entre le "correct" et "l'incorrect". D'autre part, cette notion était de nature essentiellement logique, dépendant du programme réellement testé. Notre définition, de nature formelle, est <u>intrinsèque</u>, et ne dépend du contexte d'utilisation que par l'intermédiaire des hypothèses H. Cette particularité renforce la validité du principe de couplage.

Par contre, nous ne sommes capable de construire de "bons" jeux de tests que pour des propriétés très particulières, typiquement les axiomes d'un type abstrait algébrique. Nous ne savons pas tester les propriétés temporelles, ni les propriétés spatiales existentielles.

3.5. Conclusion

Ce chapitre a décrit les principales propriétés exigibles d'un jeu de tests : essentiellement, la validité asymptotique et l'absence de biais.

Ces propriétés nous ont permis de définir rigoureusement ce que nous entendons par un "bon" jeu de tests.

Nos définitions s'inspirent largement de celles proposées par GOODENOUGH & GERHART, mais prennent en compte l'originalité de notre notion de jeu de tests : la hiérarchisation des tests sélectionnés par un critère, selon leur qualité.

A la différence de la définition proposée par nos prédécesseurs, notre notion de "bon" jeu de tests (acceptabilité) est de nature formelle, et donc intrinsèque. En particulier, elle ne dépend qu'indirectement (par l'intermédiaire des hypothèses formelles) de l'utilisation pratique du jeu de tests. En contre-partie, nous ne sommes plus capable de construire de "bons" jeux de tests pour n'importe quelle propriété. En fait, nous ne savons traiter que le cas des axiomes d'un type abstrait algébrique, à peu de choses près.

Chapitre 4 : Hiérarchies et équivalences

Au chapitre précédent, nous avons dégagé et discuté la notion d'acceptabilité. Un jeu de tests acceptable est, pour nous, la modélisation théorique d'un "bon" jeu de tests.

La plupart des auteurs qui se sont penchés sur la théorie du test, cités par [Whit 81], se sont limités à cet aspect des choses, et ont tenté, à partir d'une telle notion, de proposer une méthode, une méthodologie de construction effective d'un tel jeu de tests pour un problème donné.

Tel n'est pas notre projet.

Il nous semble que, en informatique comme ailleurs, les méthodes permettant de construire des objets complexes, non triviaux, sont généralement fondées sur la notion d'approximation successive, asymptotique éventuellement, et sur celle d'amélioration de résultats obtenus par des méthodes antérieures.

Pour être conforme à ce mouvement empiriques il nous faut donc définir ce qu'est un jeu de tests "meilleur" qu'un autre. En termes techniques, il s'agit donc d'étudier

certains ordres (partiels) sur la famille des jeux de tests d'un contexte de test donné.

Nous serons ainsi amené à définir la notion de finesse absolue, puis celle de finesse asymptotique. En fait, le principe de dualité étant d'une application ambigüe, nous définirons aussi les notions de puissance absolue et asymptotique, correspondant à celles de finesse absolue formelle, et de finesse asymptotique formelle. Nous étudierons ensuite les équivalences associées à ces ordres.

Au cours du chapitre, nous énoncerons les théorèmes de monotonie et d'equivalence montrant que ces ordres sont cohérents avec les notions dégagées au chapitre 3.

4.1. Finesse absolue.

Le problème est donc le suivant. Soit $\mathcal{C} = \langle L, S, (\mathcal{S}), A \rangle$ un contexte de tests. La famille des jeux de tests sur ce contexte forme un ensemble. Soient $\mathcal{C}$ et $\mathcal{C}'$ deux éléments de cet ensemble. Comment comparer $\mathcal{C}$ et $\mathcal{C}'$ de manière à obtenir un ordre partiel raisonnable ?

Une réponse est de poser que $\mathcal{C}$ est plus fin que $\mathcal{C}'$ si toute l'information obtenue par la réussite d'un test de $\mathcal{C}'$ peut être obtenue par la réussite du "même" test (c'est-à-dire celui de même indice) de $\mathcal{C}$. En d'autres termes, si l'on utilise $\mathcal{C}'$ au lieu de $\mathcal{C}$, l'on ne perd pas d'information.

Il nous faut noter que, dans ce raisonnement, conformément à la notion de processus de test (cf. 2.1.), nous privilégions la réussite d'un test par rapport à son échec. En fait, elle seule est réellement porteuse d'information utile (cf. chapitres 1 et 2).

4.1.1. Définitions.

Nous sommes donc conduit à la définition suivante.

<u>définition</u> : finesse absolue logique

Soit $\mathcal{C} = \langle L, S, (\mathcal{S}), A \rangle$ un contexte de test.

Soit $\mathcal{T} = \langle H, (T_n)_{n \in \mathbb{N}} \rangle$, $\mathcal{T}' = \langle H', (T'_n)_{n \in \mathbb{N}} \rangle$ deux jeux de tests sur ce contexte.

$\mathcal{T}$ est dit absolument logiquement plus fin que $\mathcal{T}'$ ($\mathcal{T} \underset{\text{abs. l.}}{\geq} \mathcal{T}'$) si

pour toute structure $\mathcal{S}$ de $(\mathcal{S})$,

et pour tout $n \in \mathbb{N}$,

si $\mathcal{S} \models T_n$, alors $\mathcal{S} \models T'_n$

Notons que cette définition eût pu être donnée dans le cas d'une famille (cf. chapitre 3), et non pas seulement d'un jeu de tests.

Conformément à notre principe, nous devons trouver un dual formel à cette définition. Nous sommes donc amené à poser :

$$H \sqcup T_n \vdash H' \sqcup T'_n.$$

Mais cette condition est insuffisante pour l'intuition. En effet, nous voulons que la finesse soit <u>réellement</u> due aux $(T_n)_{n \in \mathbb{N}}$, et non à H. Il nous faut donc contraindre H, qui, dans la pratique, n'est pas réellement informatif. Au contraire, moins les hypothèses sont fortes, meilleur est le jeu de tests.

Nous prendrons donc, confant dans notre intuition, $H' \sqcup A \vdash H \sqcup A$.

C'est à dire, sous l'hypothèse que A est valide,

les hypothèses de $\mathcal{C}$ sont moins fortes que celles de $\mathcal{C}'$.

En fait, comme nous le verrons par la suite, vu la forme très particulière des propriétés sous test (propriétés spatiales purement universelles), les hypothèses sont généralement des hypothèses d'uniformité telles que $A \vdash H$. La seconde condition est alors triviale.

Nous posons donc la définition suivante.

<u>définition</u>: finesse absolue formelle.

Soit $\mathcal{C} = \langle L, S, (\mathcal{S}), A \rangle$ un contexte de test, soit $\mathcal{T} = \langle H, (T_n)_{n \in \mathbb{N}} \rangle$, $\mathcal{T}' = \langle H', (T'_n)_{n \in \mathbb{N}} \rangle$ deux jeux de tests sur $\mathcal{C}$.

$\mathcal{T}$ est dit formellement absolument plus fin que $\mathcal{T}'$ ($\mathcal{T} \underset{abs.f.}{\geq} \mathcal{T}'$), si

$$\forall n \in \mathbb{N} \quad H \sqcup T_n \vdash H' \sqcup T'_n$$
$$H' \sqcup A \vdash H \sqcup A$$

4.1.2. Propriétés.

Tout d'abord, la dualité nous incite à énoncer la proposition suivante.

<u>proposition</u>:

La finesse absolue formelle entraîne la finesse absolue logique

En fait, seule la première propriété de la finesse absolue formelle est nécessaire, la seconde ayant un dual logique trivial, puisque, pour toute $\mathcal{S}$ de $(\mathcal{S})$, $H \models \mathcal{S}$ et $H' \models \mathcal{S}$.

La réciproque est bien sûr généralement fausse.

Enonçons maintenant les propriétés d'ordre.

<u>proposition</u>:

Soit $\mathcal{C}$ un contexte de test.

Les finesse absolues logique et formelle définissent deux ordres partiels sur les jeux de tests sur ce contexte, qui forment un ensemble.

<u>proposition</u>:

Soit $\mathcal{C}$ un contexte de test.

Selon l'ordre de finesse absolue <u>logique</u> sur les jeux de tests de $\mathcal{C}$,

il existe un plus petit élément,

deux jeux de tests admettent une borne supérieure

<u>démonstration</u>.

$\emptyset = \langle H, (T_n)_{n \in \mathbb{N}} \rangle$, avec $H = \emptyset$ et $T_n = \emptyset$ $\forall n \in \mathbb{N}$ est un jeu de tests (cf. e.4.e.1.) sur $\mathcal{C}$. Soit $\mathcal{C} = \langle H, (T_n)_{n \in \mathbb{N}} \rangle$ un autre jeu de tests sur $\mathcal{C}$.

Alors, soit n fixé, et $\mathcal{S}$ tel que $\mathcal{S} \models T_n$.
$\mathcal{S}$ valide trivialement ϕ, et donc $\mathcal{T} \underset{abs.f.}{>} \phi$.

Soit $\mathcal{T}' = \langle H', (T_n')_{n \in \mathbb{N}} \rangle$, $\mathcal{T}'' = \langle H'', (T_n'')_{n \in \mathbb{N}} \rangle$ deux $\mathcal{E}$-jeux de tests. Alors $\mathcal{T}' \sqcup \mathcal{T}'' = \langle H' \sqcup H'', (T_n' \sqcup T_n'')_{n \in \mathbb{N}} \rangle$ est un jeu de tests (cf. 2.4.2.1.) $\mathcal{E}$, et $\mathcal{T}' \sqcup \mathcal{T}'' \underset{abs.f.}{>} \mathcal{T}'$ et $\mathcal{T}' \sqcup \mathcal{T}'' \underset{abs.f.}{>} \mathcal{T}''$, puisque $T_n' \sqcup T_n'' \vdash T_n'$ et $T_n' \sqcup T_n'' \vdash T_n''$.
Soit $\mathcal{T} = \langle H, (T_n)_{n \in \mathbb{N}} \rangle$ tel que $\mathcal{T} \underset{abs.f.}{>} \mathcal{T}'$ et $\mathcal{T} \underset{abs.f.}{>} \mathcal{T}''$.
Soit n fixé, et $\mathcal{S}$ tel que $\mathcal{S} \models T_n$. Alors $\mathcal{T} \models T_n'$ et $\mathcal{S} \models T_n''$, et donc $\mathcal{S} \models T_n' \sqcup T_n''$. Donc $\mathcal{T} \underset{abs.f.}{>} \mathcal{T}' \sqcup \mathcal{T}''$.

▨

Ces propriétés ne sont pas valables, en général, pour la finesse absolue formelle. $\mathcal{T}' \sqcup \mathcal{T}''$ et $\mathcal{T}'$ ne sont, en général, pas même comparables.

Cependant, un cas particulier important est le suivant.

<u>proposition</u> :

Soit $\mathcal{E}$ un contexte de test, $\mathcal{T}' = \langle H', (T_n')_{n \in \mathbb{N}} \rangle$ et $\mathcal{T}'' = \langle H'', (T_n'')_{n \in \mathbb{N}} \rangle$ deux jeux de tests sur $\mathcal{E}$.

Supposons que H' et H'' soient A. formellement équivalents ($A \sqcup H' \vdash A \sqcup H''$).

Alors $\mathcal{T}'$ et $\mathcal{T}''$ admettent un plus petit majorant selon la finesse absolue formelle.

démonstration

Considérons $\mathcal{T}' \sqcup \mathcal{T}'' = \langle H' \sqcup H'', (T'_n \sqcup T''_n)_{n \in \mathbb{N}} \rangle$.
Nous avons $(H' \sqcup H'') \sqcup (T'_n \sqcup T''_n) \vdash H' \sqcup T'_n$.
D'autre part, $H' \sqcup A \vdash H'' \sqcup A$ et $H' \sqcup A \vdash H'$.
Donc $H' \sqcup A \vdash (H' \sqcup H'') \sqcup A$ et $\mathcal{T}' \sqcup \mathcal{T}'' \underset{abs.f.}{\geq} \mathcal{T}'$.
On conclut par symétrie.

Soit $\mathcal{T} = \langle H, (T_n)_{n \in \mathbb{N}} \rangle$ un jeu de tests sur $\mathcal{C}$, tel que $\mathcal{T} \underset{abs.f.}{\geq} \mathcal{T}'$ et $\mathcal{T} \underset{abs.f.}{\geq} \mathcal{T}''$. Alors $H \sqcup T_n \vdash H' \sqcup T'_n$ et $H \sqcup T_n \vdash H'' \sqcup T''_n$.
Donc $H \sqcup T_n \vdash (H' \sqcup H'') \sqcup (T'_n \sqcup T''_n)$. D'autre part, $H' \sqcup A \vdash H \sqcup A$ et $H'' \sqcup A \vdash H \sqcup A$. Donc $(H' \sqcup H'') \sqcup A \vdash H \sqcup A$, et $\mathcal{T} \underset{abs.f.}{\geq} \mathcal{T}' \sqcup \mathcal{T}''$.

▨

4.1.3. Théorème de monotonie

Ce théorème confirme, a posteriori, la conformité de nos définitions avec le restant de notre travail.

Théorème de monotonie (cas logique).

Soit $\mathcal{C}$ un contexte de test.

Soit $\mathcal{T}$ et $\mathcal{T}'$ deux jeux de tests sur $\mathcal{C}$.

Supposons $\mathcal{T}$ logiquement absolument plus fin que $\mathcal{T}'$. Alors,

si $\mathcal{C}'$ est logiquement asymptotiquement valide, $\mathcal{C}$ l'est;

si $\mathcal{C}$ est logiquement non biaisé, $\mathcal{C}'$ l'est.

<u>démonstration</u>.

Soit $\mathcal{C}' = \langle H', (T'_n)_{n \in \mathbb{N}} \rangle$ logiquement asymptotiquement valide. Fixons une structure $\mathcal{S}$. Considérons $\mathcal{C} = \langle H, (T_n)_{n \in \mathbb{N}} \rangle$, $\mathcal{C} \underset{\text{ass.p.}}{\geq} \mathcal{C}'$. Supposons $\mathcal{S} \models T_n$, alors $\mathcal{S} \models T'_n$, et ce, pour tout n. Donc si $\mathcal{S} \models T_n$ $\forall n \in \mathbb{N}$, $\mathcal{S} \models T'_n$ $\forall n \in \mathbb{N}$, et $\mathcal{S} \models A$. Donc $\mathcal{C}$ est logiquement asymptotiquement valide.

Supposons maintenant $\mathcal{C}$ logiquement non-biaisé. Supposons que $\mathcal{S} \models A$. Alors $\mathcal{S} \models T_n$, pour tout n. Donc $\mathcal{S} \models T'_n$ pour tout n, et $\mathcal{C}'$ est logiquement non biaisé

▨

<u>Théorème de monotonie</u> (cas formel)

Soit $\mathcal{L}$ un contexte de test, $\mathcal{C}$ et $\mathcal{C}'$ deux jeux de tests sur $\mathcal{C}$.

Supposons $\mathcal{C}$ formellement absolument plus fin que $\mathcal{C}'$. Alors,

si $\mathcal{C}'$ est formellement asymptotiquement valide, $\mathcal{C}$ l'est;

si $\mathcal{C}$ est formellement non biaisé, $\mathcal{C}'$ l'est.

<u>démonstration.</u>

Soit $\mathcal{T} = \langle L, S, (\mathcal{P}), A \rangle$, $\mathcal{T} = \langle H, (T_n)_{n \in \mathbb{N}} \rangle$, $\mathcal{T}' = \langle H', (T'_n)_{n \in \mathbb{N}} \rangle$, $\mathcal{T} \underset{abs.f.}{\succ} \mathcal{T}'$

Supposons $\mathcal{T}'$ formellement asymptotiquement valide. Etudions $H \sqcup \bigsqcup_{n \in \mathbb{N}} T_n$. Cela s'écrit $\bigsqcup_{n \in \mathbb{N}} (H \sqcup T_n)$. Mais $H \sqcup T_n \vdash H' \sqcup T'_n$. Donc cela démontre $\bigsqcup_{n \in \mathbb{N}} (H' \sqcup T'_n)$, soit $H' \bigsqcup_{n \in \mathbb{N}} T'_n$, et donc A. $\mathcal{T}$ l'est donc

Supposons $\mathcal{T}$ formellement non biaisé. Etudions $H' \sqcup A$. $H' \sqcup A \vdash H \sqcup A$ (hé oui !), et $H \sqcup A \vdash T_n$ $\forall n \in \mathbb{N}$. Donc $\mathcal{T}'$ l'est.

▨

Le choix fait lors de la définition de la finesse absolue formelle intervient donc de manière cruciale en ce qui concerne le biais, mais non pour la validité.

De manière informelle et empirique, nous pouvons donc énoncer :

"Les propriétés de validité sont croissantes, et celles d'absence de biais décroissantes".

Cette proposition est conforme avec la réciprocité des propriétés de validité et de biais. La cohérence entre le cas logique et le cas formel confirme nos définitions.

4.1.4. Exemples.

Plaçons-nous dans un contexte, et examinons le comportement de nos objets à H fixé, puis à $(T_n)_{n\in\mathbb{N}}$ fixé.

Fixons H. Alors $\mathcal{T} > \mathcal{T}'$ signifie simplement que les expériences de T_n sont plus contraignantes, logiquement ou formellement, que celles de T'_n.

Reprenons les exemples classiques : $S=\mathbb{N}$, $H=\emptyset$, $L=\{I\}$, $A=\{\forall x\ I(x)\}$. Considérons $\mathcal{T}'$ et $\mathcal{T}$ définis par

$T'_n = \{I(0) \wedge \dots \wedge I(n)\}$

$T_n = \{I(0) \wedge I(1) \wedge \dots \wedge I(2n)\}$.

De manière évidente, $\mathcal{T} \underset{\text{abs. f.}}{>} \mathcal{T}'$ (et donc $\mathcal{T} \underset{\text{abs. p.}}{>} \mathcal{T}'$).

Remarquons que cette propriété dépend étroitement des indices. Par exemple, prenons

$T_n = \{I(0) \wedge \dots \wedge I(2n)\}$ si $n \neq 0$,

$T_0 = \emptyset$.

Alors $\mathcal{T}$ et $\mathcal{T}'$ sont incomparables !

De manière plus générale, nous pouvons énoncer la proposition suivante.

<u>Proposition</u> :

Soit $\mathcal{T}$ un contexte de test, et $\mathcal{T}$ un jeu de tests, avec $\mathcal{T} = \langle H, (T_n)_{n\in\mathbb{N}} \rangle$

Il existe deux jeux de test $\mathcal{T}^+$ et $\mathcal{T}^-$ sur $\mathcal{T}$, strictement formellement plus et moins fn que $\mathcal{T}$, et l'on peut les choisir formellement non biaisé si $\mathcal{T}$ l'est, et formellement asymptotiquement valide si $\mathcal{T}$ l'est.

<u>démonstration</u>

Il suffit de prendre $\mathcal{T}^+ = \langle H^+, (T_n^+)_{n\in\mathbb{N}} \rangle$, $\mathcal{T}^- = \langle H^-, (T_n^-)_{n\in\mathbb{N}} \rangle$, avec $H^+ = H^- = H$, $T_n^+ = T_{2n}$ et $T_n^- = T_{E[n/2]}$

▨

De manière plus subtile, considérons l'exemple suivant : $S = \mathbb{N}$, $L = \{P, Q\}$, $H = \{\forall x\, (Q(x) \to P(x))\}$, $A = \{\forall x\, P(x)\}$,

$T_n' = \{P(0) \wedge \ldots \wedge P(n)\}$,

$T_n = \{Q(0) \wedge \ldots \wedge Q(n)\}$.

Nous avons : $\mathcal{T} \underset{\text{abs.f.}}{>} \mathcal{T}'$. Cependant, si $\mathcal{T}'$ est formellement non biaisé, $\mathcal{T}$ ne l'est pas.

Considérons maintenant $(T_n)_{n\in\mathbb{N}}$ fixé, et faisons varier H. Le cas logique ne présente bien sûr pas d'intérêt. Dans le cas formel,

$\mathcal{T} \underset{\text{abs.f.}}{>} \mathcal{T}'$ si $\quad H \sqcup T_n \vdash H' \sqcup T_n \quad \forall n \in \mathbb{N}$

et si $\quad H' \sqcup A \vdash H \sqcup A$

Par fabilité projective, $H \sqcup T_n \vdash H'$ pour tout $n \in \mathbb{N}$ si et seulement si $H \sqcup T_0 \vdash H'$. Si de plus $\mathcal{T}$ est sans biais, il vient $H \sqcup A \vdash T_0$. Donc

$H \cup A \vdash H \cup T_0 \vdash H'$ et $H \cup A \vdash A$. Donc $H \cup A \vdash H' \cup A$ et $H \cup A \vdash H' \cup A'$.

Si $\mathcal{T}$ et $\mathcal{T}'$ sont comparables, le plus grand étant non biaisé (il le sont alors tous les deux, par le théorème de monotonie), leurs hypothèses sont A-équivalentes. "Très peu" de jeux de tests sont donc comparables absolument.

Considérons donc $\mathcal{T}$ et $\mathcal{T}'$ tels que $H \cup A \vdash H' \cup A$. Une condition suffisante (à $(T_n)_{n \in \mathbb{N}}$ fixé) pour que $\mathcal{T} \underset{\text{abs. f.}}{>} \mathcal{T}'$ est que $H \vdash H'$. En d'autres termes, $\mathcal{T}$ est "meilleur" que $\mathcal{T}'$ si ses hypothèses sont plus fortes, quoique A-équivalentes. La réciproque est fausse.

Prenons $L = \{P\}$, $S = \mathbb{N}$, $A = \{\forall x\, P(x)\}$, $T_n = \{P(0) \wedge \ldots \wedge P(n)\}$;

$H = \{P(0) \to P(1) \wedge P(2)\}$

$H' = \{P(1)\}$.

Nous avons $H \cup T_n \vdash H' \cup T_n$, mais non $H' \cup T_0 \vdash H$; d'autre part, $A \vdash H$ et $A \vdash H'$. Donc $\mathcal{T} \underset{\text{abs. f.}}{>} \mathcal{T}'$ strictement. Mais $H \not\vdash H'$, et $H' \not\vdash H$.

Cet exemple illustre bien la complexité du comportement de la finesse absolue formelle selon H, à $(T_n)_{n \in \mathbb{N}}$ fixé

4.1.5 Conclusion

Cette partie nous a permis d'élaborer les ordres (partiels) de finesse absolue sur les jeux de tests d'un contexte donné.

Le théorème de monotonie montre que cet ordre est cohérent avec les propriétés définies au chapitre 3.

Les exemples proposés montrent qu'ils modélisent correctement l'intuition, si l'on suppose les hypothèses H fixées. Par contre, si l'on fixe $(T_n)_{n \in \mathbb{N}}$, en laissant varier que H, le comportement de la finesse absolue formelle est beaucoup plus complexe.

Le principal défaut de ces ordres est une dépendance excessive par rapport à l'indice des tests considérés. Cette dépendance est regrettable, puisque cet indice n'a de valeur que <u>relative</u>, et ne dit rien sur la <u>qualité absolue</u> d'un test.

La partie suivante est une réponse à cette critique.

4.2. Finesse asymptotique.

Comme nous l'avons vu dans la conclusion de la partie précédente, la finesse absolue est une notion globalement satisfaisante, mais privilégiant, sur un plan technique, outrageusement l'indice des tests considérés, alors que celui-ci n'a de valeur que relative. Il est en particulier regrettable que cette notion distingue $(T_n)_{n\in\mathbb{N}}$, $(T_{2n})_{n\in\mathbb{N}}$ et $(T_{E[\frac{n}{2}]})_{n\in\mathbb{N}}$ alors que, pratiquement, ces jeux de tests sont strictement identiques.

Pour résoudre ce problème, d'ordre plus technique que fondamental, nous allons appliquer une méthode classique en mathématique consistant à ne considérer que le comportement des objets "à l'infini", plutôt que sur tout leur domaine. Ainsi, par exemple, la convergence d'une série, qui ne dépend pas des "premiers" termes, mais seulement de leur évolution à l'infini.

4.2.1. Définitions

D'où les notions suivantes de finesse asymptotique :

<u>définition</u> : finesse asymptotique logique

Soit $\mathcal{C} = \langle L, S, (\mathcal{S}), A \rangle$ un contexte de test, et soit $\mathcal{T} = \langle H, (T_n)_{n \in \mathbb{N}} \rangle$ et $\mathcal{T}' = \langle H', (T'_n)_{n \in \mathbb{N}} \rangle$ deux jeux de tests sur ce contexte.

$\mathcal{T}$ est dit logiquement asymptotiquement plus fin que $\mathcal{T}'$ ($\mathcal{T} \underset{as.l.}{>} \mathcal{T}'$) si

pour tout $i \in \mathbb{N}$, il existe $j \in \mathbb{N}$ tel que pour toute structure $\mathcal{S}$ de $(\mathcal{S})$,

si $\mathcal{S} \models T_j$ alors $\mathcal{S} \models T'_i$

<u>définition</u> : finesse asymptotique formelle

Soit $\mathcal{C} = \langle L, S, (\mathcal{S}), A \rangle$ un contexte de test, et soit $\mathcal{T} = \langle H, (T_n)_{n \in \mathbb{N}} \rangle$ et $\mathcal{T}' = \langle H', (T'_n)_{n \in \mathbb{N}} \rangle$ deux jeux de tests sur ce contexte.

$\mathcal{T}$ est dit formellement asymptotiquement plus fin que $\mathcal{T}'$ ($\mathcal{T} \underset{as.f.}{>} \mathcal{T}'$) si

$$\forall i \in \mathbb{N}, \exists j \in \mathbb{N} \quad H \sqcup T_j \vdash H' \sqcup T'_i$$

et $\quad H' \sqcup A \vdash H \sqcup A$

Ces définitions sont donc directement retranscrites à partir de celles de la finesse absolue logique et formelle.

<u>4.2.2. Propriétés</u>

<u>proposition</u> :

Les finesses asymptotiques logiques et formelles définissent deux ordres partiels sur l'ensemble des jeux de tests sur un contexte donné.

<u>proposition</u> :

Soit $\mathcal{C}$ un contexte de test.
Selon l'ordre de finesse asymptotique logique sur les jeux de tests de $\mathcal{C}$,
il existe un plus petit élément,
deux jeux de tests admettent une borne supérieure.

<u>démonstration</u> :

comme en 4.1.2., $\emptyset$ est le plus petit élément.

Soit $\mathcal{C}'$ et $\mathcal{C}''$ deux jeux de tests. $\mathcal{C}' \cup \mathcal{C}''$ majore $\mathcal{C}'$ et $\mathcal{C}''$ logiquement asymptotiquement (prendre $i=j$). Soit $\mathcal{C} = \langle H, (T_n)_{n \in \mathbb{N}^*} \rangle$ tel que $\mathcal{C} \underset{as.l.}{>} \mathcal{C}'$ et $\mathcal{C} \underset{as.l.}{>} \mathcal{C}''$. Soit $i \in \mathbb{N}^*$ fixé. Il existe j' tel que, si $\mathcal{S} \models T_{j'}$ alors $\mathcal{S} \models T'_i$. Il existe j'' tel que, si $\mathcal{S} \models T_{j''}$, alors $\mathcal{S} \models T''_i$. Soit $j = \max(j', j'')$.

Par fabilité projective (en fait logique), si $\mathcal{S} \models T_j$, $\mathcal{S} \models T'_j$ et $\mathcal{S} \models T''_j$, donc $\mathcal{S} \models T'_i$

et $\mathcal{S} \models T_i''$. Donc $\mathcal{S} \models (T_i' \sqcup T_i'')$ et $\mathcal{T} \underset{as.f.}{>} \mathcal{T}' \sqcup \mathcal{T}''$.

▨

Comme en 4.1.2, cette seconde partie reste vraie dans le cas formel si les hypothèses H' et H'' sont A-équivalentes.

<u>proposition</u>:

Soit $\mathcal{T}$ un contexte de test, $\mathcal{T}' = \langle H', (T_n')_{n \in \mathbb{N}} \rangle$ et $\mathcal{T}'' = \langle H'', (T_n'')_{n \in \mathbb{N}} \rangle$ deux jeux de tests sur $\mathcal{T}$.

Supposons que H' et H'' soient A-formellement équivalentes ($H' \sqcup A \vdash H'' \sqcup A$).

Alors $\mathcal{T}'$ et $\mathcal{T}''$ admettent un plus petit majorant pour la finesse asymptotique formelle.

<u>démonstration</u>:

Considérons $\mathcal{T}' \sqcup \mathcal{T}''$. C'est un majorant de $\mathcal{T}'$ et de $\mathcal{T}''$. Soit $\mathcal{T} = \langle H, (T_n)_{n \in \mathbb{N}} \rangle$ tel que $\mathcal{T} \underset{as.f.}{>} \mathcal{T}'$ et $\mathcal{T} \underset{as.f.}{>} \mathcal{T}''$. Fixons $i \in \mathbb{N}$. Alors, il existe j' et j'' tels que

$$H \sqcup T_{j'} \vdash H' \sqcup T_i'$$

$$H \sqcup T_{j''} \vdash H'' \sqcup T_i''$$

En prenant $j = \max(j', j'')$, par fiabilité projective formelle, il vient $H \sqcup T_j \vdash (H' \sqcup H'') \sqcup (T_i' \sqcup T_i'')$.

Le reste est comme en 4.1.2.

▨

4.2.3. Finesses absolue et asymptotique

Etudions tout d'abord les rapports existant entre nos différentes définitions.

proposition :

la finesse asymptotique formelle entraîne la finesse asymptotique logique.
la finesse absolue formelle entraîne la finesse asymptotique formelle.
La finesse absolue logique entraîne la finesse asymptotique logique.

D'où, en combinant cette proposition avec celle du 4.1.2., il vient le schéma suivant.

$$\begin{tikzcd}
 & \text{finesse absolue formelle} \arrow[dl] \arrow[dr] & \\
\text{finesse absolue logique} \arrow[dr] & & \text{finesse asymptotique formelle} \arrow[dl] \\
 & \text{finesse asymptotique logique} &
\end{tikzcd}$$

Montrons par un exemple que la finesse asymptotique est plus large que la finesse absolue. Prenons $S = \mathbb{N}$, $H = \emptyset$, $A = \{\forall x\, P(x)\}$, $H' = \emptyset$;
$T_n = \{P(0) \wedge \ldots \wedge P(n)\}$,
$T'_n = \{P(0) \wedge \ldots \wedge P(n+1)\}$ pour $n \neq 0$,
$T'_0 = \emptyset$.

Par construction, $T'_0 \cup H' \nvdash T_0 \cup H$,
et pour $n > 0$, $T_n \cup H \nvdash T'_n \cup H'$.

$\mathcal{C}$ et $\mathcal{C}'$ ne sont donc pas absolument formellement comparables, ni même logiquement en général.

Cependant, ils sont formellement asymptotiquement équivalents. En effet, fixons $i \in \mathbb{N}$.

$T_i' \vdash T_i$ si $i \neq 0$

$T_1' \vdash T_0$ si $i = 0$.

et donc $\mathcal{C}' \underset{as.f.}{\succ} \mathcal{C}$.

Réciproquement, i étant toujours fixé,

$T_{i+1} \vdash T_i'$, et donc $\mathcal{C} \underset{as.f.}{\succ} \mathcal{C}'$.

Plus généralement, nous avons la caractérisation technique suivante. Définissons tout d'abord la notion de jeu de tests extrait.

<u>définition</u> : jeu de tests extrait.

Soit $\mathcal{C}$ un contexte de test, $\mathcal{C} = \langle H, (T_n)_{n \in \mathbb{N}} \rangle$ un jeu de tests sur $\mathcal{C}$.

Un jeu de tests $\mathcal{C}' = \langle H', (T_n')_{n \in \mathbb{N}} \rangle$ est dit extrait de $\mathcal{C}$ selon φ, si

φ est une application croissante (non nécessairement stricte) de $\mathbb{N}$ dans $\mathbb{N}$,

$H' = H$

$\forall n \in \mathbb{N}, \quad T_n' = T_{\varphi(n)}$

Réciproquement, étant donné un telle application φ, la donnée $\langle H, (T_{\varphi(n)})_{n \in \mathbb{N}} \rangle$ définit bien un jeu de tests sur $\mathcal{C}$.

<u>proposition</u>

Soit $\mathcal{C}$ un contexte, $\mathcal{C}$ et $\mathcal{C}'$ deux jeux de tests sur $\mathcal{C}$.

$\mathcal{C}$ est asymptotiquement logiquement plus fin que $\mathcal{C}'$, si et seulement si il existe un jeu de tests extrait de $\mathcal{C}$ qui est absolument logiquement plus fin que $\mathcal{C}'$.

<u>démonstration</u> :

Soit $\mathcal{C} = \langle H, (T_n)_{n \in \mathbb{N}} \rangle$, $\mathcal{C}' = \langle H', (T'_n)_{n \in \mathbb{N}} \rangle$.

Supposons $\mathcal{C} \underset{a.s.l.}{>} \mathcal{C}'$.

Alors pour tout i, il existe j tel que, pour toute $\mathcal{S}$, si $\mathcal{S} \models T_j$ alors $\mathcal{S} \models T'_i$.

Choisissons un tel j (le plus petit par exemple), et posons $j = f(i)$.

Prenons alors $\varphi(i) = \sup(f(0), \dots, f(i))$

φ est bien croissante de $\mathbb{N}$ dans $\mathbb{N}$, et si $\mathcal{S} \models T_{\varphi(i)}$, par fiabilité projective, $\mathcal{S} \models T_{f(i)}$ et donc $\mathcal{S} \models T'_i$.

Réciproquement, soit φ tel que le jeu de tests extrait de $\mathcal{C}$ selon φ soit absolument plus fin logiquement que $\mathcal{C}'$. Alors, fixons i ; pour toute $\mathcal{S}$, si $\mathcal{S} \models T_{\varphi(i)}$, alors $\mathcal{S} \models T'_i$. Il suffit de prendre $j = \varphi(i)$, qui ne dépend pas du choix de $\mathcal{S}$.

▨

Par un raisonnement strictement analogue, nous pouvons énoncer cette proposition.

Proposition :

Soit $\mathcal{C}$ un contexte, $\mathcal{T}$ et $\mathcal{T}'$ deux jeux de tests sur $\mathcal{C}$.

$\mathcal{T}$ est asymptotiquement formellement plus fin que $\mathcal{T}'$, si et seulement si il existe un jeu de tests extrait de $\mathcal{T}$ qui est absolument formellement plus fin que $\mathcal{T}'$.

4.2.4. Théorème de monotonie

Ce théorème, énoncé en 4.1.3., reste valable dans le cas asymptotique.

Théorème de monotonie (cas logique)

Soit $\mathcal{C}$ un contexte de test, $\mathcal{T}$ et $\mathcal{T}'$ deux tests sur $\mathcal{C}$.

Supposons $\mathcal{T}$ logiquement asymptotiquement plus fin que $\mathcal{T}'$. Alors,

si $\mathcal{T}'$ est logiquement asymptotiquement valide, $\mathcal{T}$ l'est ;

si $\mathcal{T}$ est logiquement non-biaisé, $\mathcal{T}'$ l'est.

démonstration :

Posons $\mathcal{C} = \langle L, S, (\mathcal{S}), A \rangle$, $\mathcal{T} = \langle H, (T_n)_{n \in \mathbb{N}} \rangle$, $\mathcal{T}' = \langle H', (T'_n)_{n \in \mathbb{N}} \rangle$.

Supposons $\mathcal{T}'$ logiquement asymptotiquement valide, et considérons $\mathcal{S}$ telle que $\mathcal{S} \models T_n$ $\forall n \in \mathbb{N}$.

Fixons i, et considérons le j associé. Puisque $\mathcal{S} \models T_j$, $\mathcal{S} \models T'_i$, et ce pour tout $i \in \mathbb{N}$. Donc $\mathcal{S} \models A$.

Supposons $\mathcal{T}$ logiquement non biaisé, et considérons $\mathcal{S}$ telle que $\mathcal{S} \models A$. Alors $\mathcal{S} \models T_n$ $\forall n \in \mathbb{N}$. Fixons i et considérons le j associé. $\mathcal{S} \models T_j$, donc $\mathcal{S} \models T'_i$ et ce pour $i \in \mathbb{N}$ quelconque.

▨

De manière analogue, énonçons le cas formel.

<u>Théorème de monotonie</u> (cas formel)

Soit $\mathcal{C}$ un contexte de test, $\mathcal{T}$ et $\mathcal{T}'$ deux tests sur $\mathcal{C}$.

Supposons $\mathcal{T}$ formellement asymptotiquement plus fin que $\mathcal{T}'$.

Si $\mathcal{T}'$ est formellement asymptotiquement valide, $\mathcal{T}$ l'est.

Si $\mathcal{T}$ est formellement non biaisé, $\mathcal{T}'$ l'est.

<u>démonstration</u> :

Posons $\mathcal{C} = \langle L, S, (\mathcal{S}), A \rangle$, $\mathcal{T} = \langle H, (T_n)_{n \in \mathbb{N}} \rangle$, $\mathcal{T}' = \langle H', (T'_n)_{n \in \mathbb{N}} \rangle$.

Supposons $\mathcal{T}'$ formellement asymptotiquement valide. Examinons $H \sqcup \bigsqcup_{n \in \mathbb{N}} T_n$. Fixons i. Il existe j tel que $H \sqcup T_j \vdash H' \sqcup T'_i$. Donc $H \sqcup \bigsqcup_{n \in \mathbb{N}} T_n \vdash H' \sqcup T'_i$, et ce pour tout $i \in \mathbb{N}$. Donc $H \sqcup \bigsqcup_{n \in \mathbb{N}} T_n \vdash H' \sqcup \bigsqcup_{n \in \mathbb{N}} T'_n$, qui démontre A. Donc $H \sqcup \bigsqcup_{n \in \mathbb{N}} T_n \vdash A$.

Supposons $\mathcal{T}$ formellement non biaisé. Examinons $H' \sqcup A$, $H' \sqcup A \vdash H \sqcup A$, et $H \sqcup A \vdash T_n$ $\forall n \in \mathbb{N}$. Donc $H' \sqcup A \vdash H \sqcup T_n$, et ce quel que soit n. Comme ci-dessus, fixons i et prenons le j associé ; $H \sqcup T_j \vdash H' \sqcup T'_i$, et donc $H' \sqcup A \vdash H' \sqcup T'_i$ $\forall i \in \mathbb{N}$. Donc $H' \sqcup A \vdash T'_i$ $\forall i \in \mathbb{N}$.

▨

Les remarques concluant 4.1.3. restent donc valables dans ce cas. Remarquons que, par les implications décrites en 4.2.3., ces deux théorèmes entraînent les deux précédents.

4.2.5 Conclusion

Pour répondre aux critiques énoncées dans la première partie, nous avons proposé une amélioration de nos définitions permettant de ne considérer que le comportement asymptotique des indices des tests.

Cette amélioration nous conduit à des définitions strictement plus large que les précédentes, sans que les résultats fondamentaux, les théorèmes de monotonie notamment, en soit perturbés.

Par contre, si la finesse asymptotique logique paraît une notion tout à fait satisfaisante, sa duale formelle est encore très restrictive, et ne permet de comparer que "peu" de jeux de tests dont les hypothèses ne soient pas A-formellement équivalentes.

4.3. Puissance

Nous présentons, dans cette partie, une variante des notions formelles de finesse absolue et de finesse asymptotique permettant de répondre à la critique exposée précédemment.

En définissant la notion de finesse absolue formelle (cf. 4.1.1.), nous avons exprimé notre souci du fait qu'un jeu de test ne doit pas être meilleur qu'un autre du simple fait de ses hypothèses. Au contraire, nous avons essayé (maladroitement peut-être, ou trop brutalement) de définir un ordre qui privilégie les jeux de tests construits sur des hypothèses faibles.

Cette démarche nous a conduit aux notions de finesse absolue et asymptotique formelles, où deux jeu de tests ne sont comparables que si, en pratique, leurs hypothèses sont A-équivalentes formellement.

Pour éviter cette dégénérescence, nous proposons la notion de puissance (qui n'a aucun rapport avec la puissance d'un test statistique) absolue et asymptotique.

En fait, nous verrons que cette tentative n'est guère plus satisfaisante.

4.3.1. Définitions

Nous proposons donc la variante suivante pour la définition de la finesse asymptotique formelle.

<u>définition</u> : puissance

Soit $\mathcal{C}$ un contexte de test, $\mathcal{C} = \langle H, (T_n)_{n \in \mathbb{N}} \rangle$ et $\mathcal{C}' = \langle H', (T'_n)_{n \in \mathbb{N}} \rangle$ deux jeux de tests sur ce contexte.

$\mathcal{C}$ sera dit (asymptotiquement) plus puissant que $\mathcal{C}'$ ($\mathcal{C} \underset{puiss.}{>} \mathcal{C}'$) si

$\forall i \in \mathbb{N}, \exists j \in \mathbb{N}$

$$H \sqcup T_j \vdash H' \sqcup T'_i$$

et $$H \sqcup A \vdash H' \sqcup A$$

La définition de la puissance absolue est obtenue en imposant $i = j$.

Remarquons que la duale logique de cette définition est aussi celle de la finesse asymptotique logique.

4.3.2. Propriétés

<u>proposition</u>:

La puissance définit un ordre partiel sur

l'ensemble des jeux de tests d'un contexte.
Cet ordre admet un plus petit élément.
Deux éléments admettent un plus petit majorant.

<u>démonstration</u> :

$\emptyset$ est bien sûr le plus petit élément.
$\mathcal{T}' \sqcup \mathcal{T}''$ est le plus petit majorant de $\mathcal{T}'$ et $\mathcal{T}''$, puisque $(H' \sqcup H'') \sqcup A \vdash H' \sqcup A$ et $(H' \cup H'') \sqcup A \vdash H'' \sqcup A$.

Le rapport avec les définitions précédentes s'établit ainsi.

<u>proposition</u> :

La puissance (asymptotique) entraîne la finesse asymptotique logique.
Soit $\mathcal{T} = \langle H, (T_n)_{n \in \mathbb{N}} \rangle$, $\mathcal{T}' = \langle H, (T'_n)_{n \in \mathbb{N}} \rangle$ deux jeux de tests d'un contexte $\mathcal{C} = \langle L, S, (\mathcal{S}), A \rangle$.

Supposons $H \sqcup A \dashv H' \sqcup A$ (A. equivalence formelle).

Alors $\mathcal{T}$ est formellement asymptotiquement plus fin que $\mathcal{T}'$, si et seulement si il est plus puissant que lui.

Le théorème de monotonie n'est pas conservé en général. Seule la croissance de la validi.

té asymptotique formelle reste vraie. Pour le cas du biais, le contre-exemple suivant montre que le théorème n'est pas vrai en général.

Considérons $A = \{\forall x\ P(x)\}$, $L = \{P, Q\}$, $S = \mathbb{N}$;
Prenons $\mathcal{C} = \langle H, (T_n)_{n \in \mathbb{N}} \rangle$ avec

$H = \{\forall x\ (P(x) \to Q(x))\}$

$T_n = \{Q(0) \wedge P(0) \wedge \dots \wedge P(n)\}$

et $\mathcal{C}' = \langle H', (T'_n)_{n \in \mathbb{N}} \rangle$ avec

$H' = \emptyset$

$T'_n = T_n$.

Puisque $H \vdash H'$ et $T_n = T'_n$, $\mathcal{C} \underset{\text{puiss.}}{>} \mathcal{C}'$.
$\mathcal{C}$ est non biaisé formellement, puisque $A \cup H \vdash T_n$. Mais $\mathcal{C}'$ ne l'est pas, puisque $\forall x\ P(x) \not\vdash Q(0)$. $\mathcal{C}$ et $\mathcal{C}'$ sont de plus incomparables selon la finesse asymptotique formelle.

4.3.3. Conclusion.

Le théorème de monotonie n'étant pas vérifié, malgré le bon comportement de l'ordre défini, cette variante n'a, à notre sens, qu'un intérêt "bourbakiste".

De plus, dans tous les cas pratiques ou intéressants, l'on a, en fait, H et H' A-formellement équivalentes, et cette notion se confond avec la finesse asymptotique formelle. C'est notamment le cas de

l'équivalence asymptotique formelle. Nous pouvons donc énoncer la proposition suivante, qui justifie que nous laissions, par la suite, la notion de puissance dans l'ombre.

<u>proposition</u>:

Soit $\mathcal{T}$ et $\mathcal{T}'$ deux jeux de tests sur un contexte $\mathcal{C}$.

$\mathcal{T}$ et $\mathcal{T}'$ sont équivalents selon la puissance (asymptotique), si et seulement si ils le sont selon la finesse asymptotique formelle.

4.4. Equivalences

Les ordres que nous avons étudiés, jusqu'à présent, sur l'ensemble des jeux de tests d'un contexte ne sont que des ordres partiels, et, en fait, relativement pauvres. Ils ne nous seront donc, pratiquement, que de peu d'utilité.

Par contre, à tout ordre partiel, l'on peut associer canoniquement une relation d'équivalence qui identifie tous les éléments x et y vérifiant $x > y$ et $y > x$. Cette relation produit un ensemble quotient, et l'ordre induit sur ce quotient un ordre partiel antisymétrique.

Nous allons donc étudier dans cette partie les équivalences déduites des ordre étudiés précédemment. Le 4.3.3. montre que celle déduite de la finesse asymptotique formelle s'identifie à celle déduite de la puissance (asymptotique). Comme nous allons le voir, elles donnent lieu à des propriétés extrêmement intéressantes, notamment au théorème d'équivalence qui énonce que, en un certain sens, il n'existe qu'un seul "bon" jeu de tests pour un contexte donné.

4.4.1. Propriétés

Des propriétés énoncées dans les parties précédentes, nous déduisons immédiatement l'énoncé suivant.

proposition :

L'équivalence asymptotique formelle entraîne l'équivalence asymptotique logique.
L'equivalence absolue formelle entraîne l'équivalence absolue logique.
L'équivalence asymptotique formelle entraîne l'equivalence absolue formelle.
L'équivalence asymptotique logique entraîne l'equivalence absolue logique

Nous résumons ceci par le schema suivant

$$\begin{tikzcd}
 & \text{equiv. abs. } f. \arrow[dl] \arrow[dr] & & \\
\text{equiv. abs. } l. \arrow[dr] & & \text{equiv. as. } f \arrow[dl] \arrow[r, leftrightarrow] & \begin{array}{c}\text{equivalence}\\ \text{de puissance}\\ \text{(asymptotique)}\end{array} \\
 & \text{equiv. as. } l &
\end{tikzcd}$$

La propriété suivante justifie, a posteriori, la définition de la finesse asymptotique formelle. Définissons tout d'abord la notion de jeu de tests strictement extrait d'un autre.

<u>définition</u> : jeu de tests strictement extrait

Soit $\mathcal{T}$ un jeu de tests sur un contexte $\mathcal{C}$.
Soit $\mathcal{T}'$ un jeu de tests (sur $\mathcal{C}$) extrait de $\mathcal{T}$ selon φ.
$\mathcal{T}'$ est dit strictement extrait de $\mathcal{T}$ si φ est d'image infinie ($|\varphi(\mathbb{N})| = \infty$)

De manière équivalente (φ étant croissante), l'on peut imposer $\lim_{n \to \infty} \varphi(n) = \infty$.

Nous pouvons alors énoncer la proposition suivante.

<u>proposition</u> :

Soit $\mathcal{T}$ un jeu de tests sur un contexte $\mathcal{C}$;
$\mathcal{T}$ est formellement asymptotiquement équivalent à tout jeu de tests strictement extrait de $\mathcal{T}$.

<u>démonstration</u> :

Il suffit de montrer que :

$\forall m \; \exists n \; T'_n \sqcup H \vdash T_m$

et $\forall m \; \exists n \; T_n \sqcup H \vdash T'_m$.

Fixons m. φ étant d'image infinie, cette image contient un $m_0 > m$. Soit n tel que $\varphi(n) = m_0$. Alors $T'_n \sqcup H = T_{m_0} \sqcup H$, et, par projective fiabilité, cela démontre T_m.
Donc $T'_n \sqcup H = T_m$.

Dans le deuxième cas, fixons m.

par définition, $T'_m = T_{\varphi(m)}$. Il suffit de prendre $n = \varphi(m)$.

▨

Cette proposition nous montre clairement que, vis à vis de cette équivalence, <u>la notion d'indice perd toute valeur intrinsèque</u>. L'indicage des tests d'un jeu de tests n'a de valeur que <u>relative</u>, technique, mais ne peut en aucune façon servir de base à la définition d'une notion absolue et intrinsèque de <u>qualité</u> d'un test pour un problème donné.

4.4.2 Théorème de stabilité

Une conséquence immédiate du théorème de monotonie (cf. 4.2.4.) est le théorème de stabilité, qui s'énonce de la manière suivante.

<u>Théorème de stabilité</u> (cas logique)

Un jeu de tests logiquement asymptotiquement équivalent à un jeu de tests logiquement acceptable l'est aussi.

<u>Théorème de stabilité</u> (cas formel)

Un jeu de tests formellement asymptotiquement équivalent à un jeu de tests (formellement) acceptable l'est aussi

Ces deux parties du théorème justifient, a posteriori, la notion d'acceptabilité, définie au paragraphe 3.4., vis à vis de celle d'équivalence asymptotique.

Elles nous montrent aussi que le problème de la construction de jeux de tests sur un contexte se ramène à celui de l'étude des classes de jeux de tests selon l'équivalence asymptotique, et de représentants privilégiés de ces classes.

4.4.3. Théorème d'équivalence

Contre toute attente, <u>le cas formel</u> du théorème de stabilité admet une réciproque (partielle).

<u>Théorème d'équivalence</u>

Soit $\mathcal{C} = \langle L, S, (\mathcal{S}), A \rangle$ un contexte de test;

Soit $\mathcal{T} = \langle H, (T_n)_{n\in\mathbb{N}} \rangle$ et $\mathcal{T}' = \langle H', (T'_n)_{n\in\mathbb{N}} \rangle$ deux jeux de tests sur ce contexte

Supposons qu'il existe une sous-théorie A_0 de A, n'ayant qu'<u>un nombre fini d'axiomes non-logiques</u>, telle que :

$$A_0 \sqcup H \vdash A_0 \sqcup H'.$$

Alors, si $\mathcal{T}$ et $\mathcal{T}'$ sont formellement acceptables, ils sont formellement asymptotiquement équivalents.

<u>démonstration</u>

$\mathcal{T}$ et $\mathcal{T}'$ sont formellement acceptables, donc

$$H \sqcup_{n\in\mathbb{N}} T_n \vdash H \sqcup A$$

$$H' \sqcup_{n\in\mathbb{N}} T'_n \vdash H' \sqcup A$$

Etudions $H' \sqcup T'_i$, pour $i \in \mathbb{N}$ fixé.

Etudions T'_i. Nous avons

$$H \sqcup_n T_n \vdash H \sqcup A \vdash H \sqcup A_0 \vdash H' \sqcup A_0 ,$$

et aussi $H \sqcup_n T_n \vdash H \sqcup A \vdash A$.

Donc $H \sqcup_n T_n \vdash H' \sqcup A_0 \sqcup A$,

$$H \sqcup_n T_n \vdash H' \sqcup A \vdash H' \sqcup_n T'_n \vdash T'_i$$

Donc $H \sqcup_n T_n \vdash T'_i$.

Mais T'_i ne contenant qu'un nombre fini d'axiomes non-logiques, il existe j_1 tel que $H \sqcup_{n\leq j_1} T_n \vdash T'_i$,

et par fiabilité projective, $H \sqcup_{n\leq j_1} T_n \vdash H \sqcup T_{j_1}$

Donc $H \sqcup T_{j_1} \vdash T'_i$.

Etudions maintenant H'.

$$H \bigsqcup_n T_n \vdash H \sqcup A \vdash H \sqcup A_0$$

Donc $H \bigsqcup_n T_n \vdash A_0$

Mais A_0 n'ayant qu'un nombre fini d'axiomes non-logiques, par le même raisonnement que précédemment (compacité puis fiabilité), il existe j_2 tel que

$$H \sqcup T_{j_2} \vdash A_0$$

Donc $H \sqcup T_{j_2} \vdash A_0 \sqcup H \vdash A_0 \sqcup H' \vdash H'$

et $H \sqcup T_{j_2} \vdash H'$.

Prenons $j = \max(j_1, j_2)$.

Par fiabilité projective,

$$H \sqcup T_j \vdash H'$$

$$H \sqcup T_j \vdash T'_i$$

Donc $H \sqcup T_j \vdash H' \sqcup T'_i$, et pour tout $i \in \mathbb{N}$ il existe un tel j.

De $A_0 \sqcup H' \vdash A_0 \sqcup H$, nous déduisons

$$A \sqcup H' \vdash A_0 \sqcup H.$$

Donc $A \sqcup H' \vdash H$, et $A \sqcup H' \vdash A$.

Donc $A \sqcup H' \vdash A \sqcup H$

Donc $\mathcal{T} \underset{as.f.}{\geq} \mathcal{T}'$, et par symétrie, $\mathcal{T}$ et $\mathcal{T}'$ sont formellement asymptotiquement équivalents.

▨

Intuitivement, ce théorème signifie que si $\mathcal{C}$ et $\mathcal{C}'$ sont acceptables et "assez proches", alors ils sont équivalents. Autrement dit, deux jeux de tests acceptables sur un contexte donné sont soit équivalents, soit très "éloignés".

L'exemple suivant nous montre deux jeux de tests ainsi "éloignés".

Prenons $L = \{P\}$, $S = \mathbb{N}$, $A = \{P(0), P(1), \dots\}$

Considérons

$\mathcal{C} = \langle H, (T_n)_{n \in \mathbb{N}} \rangle$ avec

$H = \emptyset$, $T_n = \{P(0) \wedge \dots \wedge P(n)\}$,

$\mathcal{C}' = \langle H', (T'_n)_{n \in \mathbb{N}} \rangle$ avec

$H' = A$, $T_n = \emptyset$

$\mathcal{C}$ est acceptable, car $\bigsqcup_{n \in \mathbb{N}} T_n \vdash A$

$\mathcal{C}'$ l'est aussi ; la condition $H \bigsqcup_n T_n \vdash H \sqcup A$ s'écrit en effet $A \bigsqcup_n \emptyset \vdash A \sqcup A$

S'ils étaient équivalents asymptotiquement formellement, nous aurions $\mathcal{C} \underset{as.f.}{\geq} \mathcal{C}'$; pour tout i, il existe j tel que $H \sqcup T_j \vdash H' \sqcup T'_i$.

Fixons i. Il nous faut trouver j tel que

$$\emptyset \sqcup \{P(0) \wedge \dots \wedge P(j)\} \vdash \{P(0), P(1), \dots\} \sqcup \emptyset$$

– ce qui est bien évidemment impossible !

Vérifions que $\mathcal{C}$ et $\mathcal{C}'$ ne vérifient pas les hypothèses du théorème précédent. Il nous faudrait une sous-théorie A_0 de A telle que :

$$A_0 \sqcup H \vdash A_0 \sqcup H'$$

Ceci s'écrit, puisque $A_0 \subseteq A$

$$A_0 \models A$$

Mais A_0 étant une sous-théorie de A, ses axiomes non-logiques sont de la forme $P(n)$.
Etant en nombre fini, il existe n_0 tel que

$$\{P(0)P(1)\ldots P(n_0)\} \vdash A_0$$

Mais $A \vdash \{P(0)\ldots P(n_0)\}$.

Donc $A \models \{P(0)\ldots P(n_0)\}$; contradiction.

Cependant, $H \cup A \models H' \cup A$

La proposition suivante montre que de tels cas sont cependant rarissimes.

<u>proposition</u>

Soit $\mathcal{C} = \langle L, S, (\mathcal{S}), A \rangle$ un contexte de test, et soit $\mathcal{T} = \langle H, (T_n)_{n \in \mathbb{N}} \rangle$, $\mathcal{T}' = \langle H', (T'_n)_{n \in \mathbb{N}} \rangle$ deux jeux de tests sur ce contexte

Supposons que $\mathcal{T}$ et $\mathcal{T}'$ sont acceptables sur $\mathcal{C}$, et <u>comparables</u> pour la finesse asymptotique formelle.

Si $\mathcal{T}$ et $\mathcal{T}'$ sont <u>finiment asymptotiquement valides</u>, alors il existe une sous-théorie A_0 de A n'ayant qu'un nombre fini d'axiomes non-logiques, telle que

$$A_0 \cup H \models A_0 \cup H'$$

<u>démonstration</u>

$\mathcal{T}$ et $\mathcal{T}'$ sont acceptables.

Donc, considérant $\mathcal{T}$ par exemple,

$$H \sqcup \bigcup_n T_n \models H \sqcup A. \qquad (1)$$

Ils sont finiment acceptables.

Donc, considérant $\mathcal{T}$, il existe n_0 tel que

$$H \sqcup T_{n_0} \vdash A \qquad (2)$$

De (1) et (2) il vient $H \sqcup T_{n_0} \models H \sqcup A \qquad (3)$

De (1), on déduit

$$H \sqcup A \vdash T_{n_0} \qquad (4)$$

Par compacité, puisque T_{n_0} n'a qu'un nombre fini d'axiomes non logiques, il existe B, sous-théorie de A n'ayant qu'un nombre fini d'axiomes non-logiques, telle que

$$H \sqcup B \vdash T_{n_0} \qquad (5)$$

D'où $H \sqcup B \vdash H \sqcup T_{n_0} \vdash H \sqcup A \vdash H \sqcup B$

et $H \sqcup T_{n_0} \models H \sqcup B \qquad (6)$

De même pour $\mathcal{T}'$, on peut trouver n'_0 et B' tels que $H' \sqcup T'_{n'_0} \models H' \sqcup B' \qquad (7)$

Prenons $A_0 = B \sqcup B'$. A_0 répond aux exigences de la proposition, et

$$H \sqcup \bigcup_n T_n \models H \sqcup T_{n_0} \models H \sqcup A_0 \qquad (8)$$

$$H' \sqcup \bigcup_n T'_n \models H' \sqcup T'_{n'_0} \models H' \sqcup A_0 \qquad (9).$$

Utilisons maintenant le fait que $\mathcal{T}$ et $\mathcal{T}'$ sont comparables. Supposons $\mathcal{T} \geq_{\text{a.f.}} \mathcal{T}'$

Nous avons $H' \sqcup A \vdash H \sqcup A. \qquad (10)$

Pour tout i, il existe j tel que

$$H \sqcup T_j \vdash H' \sqcup T'_i \qquad (11).$$

Donc, pour tout i,

$$H \sqcup_n T_n \vdash H' \sqcup T'_i$$

et donc

$$H \sqcup_n T_n \vdash H' \sqcup T'_n. \qquad (12)$$

Mais $\mathcal{T}$ et $\mathcal{T}'$ étant acceptables,

$$H \sqcup_n T_n \dashv\vdash H \sqcup A$$

$$H' \sqcup_n T'_n \dashv\vdash H' \sqcup A$$

et de (10) l'on déduit

$$H' \sqcup_n T'_n \vdash H \sqcup T_n \qquad (13)$$

et donc, par (12) et (13)

$$H' \sqcup_n T'_n \dashv\vdash H \sqcup_n T_n \qquad (14)$$

et de (14), (8) et (9)

$$H \sqcup A_0 \dashv\vdash H' \sqcup A_0.$$

▨

Remarquons au passage que (14) est vrai, même si $\mathcal{T}$ et $\mathcal{T}'$ ne sont pas finiment formellement valides. L'on peut alors conclure que

$$H \sqcup A \dashv\vdash H' \sqcup A ;$$

mais une telle équivalence n'est en général pas vérifiée avec une sous-théorie A_0 de A.

Dans l'exemple précédent, nous avions en effet $\mathcal{T}' \underset{\text{as. f.}}{>} \mathcal{T}$; $\mathcal{T}'$ et $\mathcal{T}$ était donc comparables.

Cette proposition nous permet maintenant d'énoncer le "théorème fondamental du test" qui exprime qu'en un certain sens, il n'existe, sur un contexte donné, qu'au plus un seul "bon" jeu de tests.

<u>Théorème fondamental du test</u>

Soit $\mathcal{T}$ et $\mathcal{T}'$ deux jeux de tests acceptables sur un contexte $\mathcal{C}$.

Supposons $\mathcal{T}$ et $\mathcal{T}'$ finiment formellement valides. Si $\mathcal{T}$ et $\mathcal{T}'$ sont comparables (selon l'ordre de la finesse asymptotique formelle), ils sont équivalents.

Le théorème de compacité (cf. 3.2.4) permet un énoncé plus précis applicable dans la plupart des cas concrets.

<u>Corollaire</u>

Soit $\mathcal{C} = \langle L, S, (\mathcal{S}), A \rangle$ un contexte de test, A n'ayant qu'<u>un nombre fini d'axiomes non-logiques</u>. Soit $\mathcal{T}$ et $\mathcal{T}'$ deux jeux de tests acceptables sur $\mathcal{C}$. Si $\mathcal{T}$ et $\mathcal{T}'$ sont comparables (selon l'ordre de la finesse asymptotique formelle), ils sont équivalents

Autrement dit, si nous avons pu construire un jeu de tests acceptable sur un tel contexte, il n'en existe pas de strictement plus fn.

Nous sommes donc assuré d'avoir exhibé, au sens de la finesse asymptotique formelle, le "meilleur" jeu de tests possible.

4.4.4. Discussion

Il est bien évident que ces théorèmes sont appelés à jouer des rôles essentiels dans toute stratégie concrète de production de jeux de tests.

En effet, ils assurent que, quelle que soit la stratégie employée, si celle-ci permet de trouver un jeu de tests acceptable, ce jeu de tests est maximal pour l'ordre de la finesse asymptotique formelle.

Ceci ne veut bien sûr pas dire qu'il n'en existe pas d'autre. Le problème de l'optimisation consiste précisément à comparer, selon des critères extérieurs à la théorie, le jeu de tests trouvé avec ceux qui lui sont formellement asymptotiquement équivalents, ou avec d'autres jeux de tests acceptables, mais incomparables pour l'ordre utilisé. Ce problème difficile sera discuté largement dans les chapitres 5, 6 et 7.

Intuitivement, le théorème se justifie par le fait que la validité et le biais croissent en sens contraires, et que l'acceptabilité correspond précisément à la conjonction de ces deux propriétés. Cependant, cette explication ne suffit pas à montrer que <u>ce théorème n'a pas d'équivalent logique</u>, comme le montre le contre-exemple suivant.

Prenons $S = \mathbb{N}$, $L = \{P, Q\}$, $A = \{\forall x\, P(x)\}$; prenons pour $(\mathcal{S})$ la famille des L-structures sur $\mathbb{N}$ telles que P et Q soient d'interprétation calculables, et validant $H = \{(\forall x\, P(x)) \leftrightarrow (\forall x\, Q(x))\}$

Prenons alors $T_n = \{P(0) \wedge \ldots \wedge P(n)\}$, et $T'_n = \{Q(0) \wedge \ldots \wedge Q(n)\}$.

Soit $\mathcal{T} = \langle H, (T_n)_{n \in \mathbb{N}} \rangle$ et $\mathcal{T}' = \langle H, (T'_n)_{n \in \mathbb{N}} \rangle$

$\mathcal{T}$ est logiquement acceptable ; en effet, $\mathcal{S} \models \forall x\, P(x)$ si et seulement si $\mathcal{S} \models P(0) \wedge \dots \wedge P(n)$ pour tout n, puisque $S = \mathbb{N}$.

$\mathcal{T}'$ est logiquement acceptable ; fixons $\mathcal{S}$ dans la famille $(\mathcal{S})$; si $\mathcal{S} \models A$, $\mathcal{S} \models \forall x\, P(x)$ et donc $\mathcal{S} \models \forall x\, Q(x)$, et $\mathcal{S} \models T'_n$ $\forall n \in \mathbb{N}$; si $\mathcal{S} \models T'_n$ $\forall n \in \mathbb{N}$, $\mathcal{S} \models \forall x\, Q(x)$, $\mathcal{S} \models \forall x\, P(x)$, et $\mathcal{S} \models A$.

Supposons $\mathcal{T} \succ \mathcal{T}'$. Considérons T'_0. Il existe n_0 tel que si $\mathcal{S} \models T_{n_0}$, alors $\mathcal{S} \models T'_0$; en d'autres termes, si $P_{\mathcal{S}}(0), \dots, P_{\mathcal{S}}(n_0)$, alors $Q_{\mathcal{S}}(0)$. Il suffit de prendre $\mathcal{S}$ telle que $Q_{\mathcal{S}}(x) = (x \neq 0)$ et $P_{\mathcal{S}}(x) = (x \leq n_0)$, qui valide bien H.

Montrons, par un contre-exemple, que l'hypothèse sur H et H' est absolument nécessaire.

Prenons $\cancel{\mathcal{S}}$ $S = \mathbb{N}$, $L = \{P\}$, $A = \{P(0), P(1), \dots\}$, $T_n = T'_n = \{P(0) \wedge \dots \wedge P(n)\}$.

Considérons :

$$H = \{P(0) \wedge \dots \wedge P(k) \rightarrow \forall x\, P(x)\}$$

$$H' = \{P(0) \wedge \dots \wedge P(k) \rightarrow P(i), \text{ pour tous les } i \geq 0\}$$

Soit $\mathcal{T} = \langle H, (T_n)_{n \in \mathbb{N}} \rangle$ et $\mathcal{T}' = \langle H', (T'_n)_{n \in \mathbb{N}} \rangle$.

Etudions $\mathcal{T}$. $\mathcal{T}$ est formellement non biaisé, puisque $A \sqcup H \vdash T_n$ $\forall n \in \mathbb{N}$. Il est formellement asymptotiquement valide puisque $H \sqcup T_k \vdash A$ (validité finie formelle, cf. 3.2.4.). $\mathcal{T}$ est donc acceptable.

Etudions $\mathcal{T}'$. $\mathcal{T}'$ est formellement non

biaisé car $A \cup H \vdash P(i)$ pour tout $i \geq 0$, et donc $A \cup H \vdash T_n'$ $\forall n \in \mathbb{N}$. $\mathcal{C}'$ est formellement asymptotiquement valide puisque $H \cup T_k \vdash A$. $\mathcal{C}'$ est donc (formellement) acceptable.

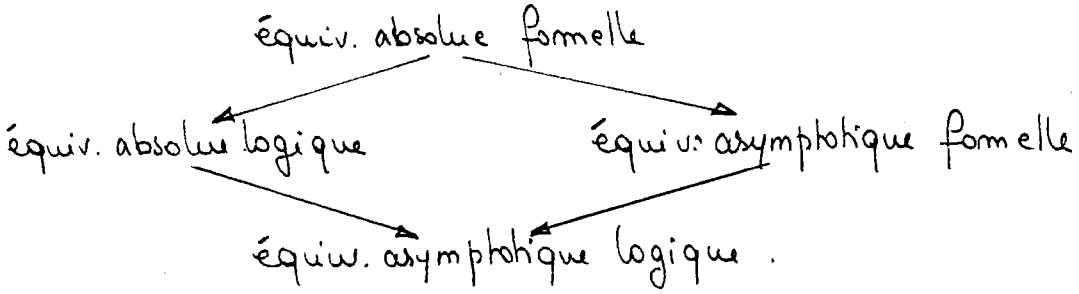
Pour montrer qu'ils ne sont pas équivalents, étudions $A \cup H'$ et $A \cup H$. $A \cup H' \vdash A$, $A \cup H \vdash \{\forall x P(x)\}$, et il est désormais classique (cf. annexe 3) que $\{P(0), P(1) \ldots\}$ n'est pas formellement equivalent à $\{\forall x P(x)\}$.

Cet exemple rejoint donc celui présenté en 4.4.3., mais les deux jeux de tests présentés ici sont "plus éloignés" qu'alors.

4.4.5 Conclusion

Cette partie nous a conduit à étudier les équivalences déduites des ordres développés précédemment sur les jeux de tests d'un contexte.

Nous avons montré que celles déduites de la notion de puissance s'identifient à celles déduites de celle de finesse. Celles-ci se disposent selon le schéma suivant.



Parmi ces différentes notions, l'équivalence asymptotique formelle est la plus fructueuse. Le théorème de stabilité montre qu'elle conserve l'acceptabilité ; celui d'équivalence montre que deux jeux de tests acceptables comparables sont en fait équivalents. Selon cette équivalence, il n'existe, en un certain sens, qu'un seul jeu de tests acceptable sur un contexte donné. De manière plus précise, un tel jeu de tests ayant été trouvé, il n'en existe pas de meilleur.

D'autre part, deux jeux de tests ne différant que par l'indiçage de leur famille de tests sont équivalents. Ce résultat nous enlève donc

tout espoir de fonder une théorie de la qualité d'un test intrinsèque sur l'indice de ce test dans le jeu de tests considéré. Cet indice, même si l'on se restreint aux jeux de tests acceptable, n'a de valeur que relative.

4.5. Conclusion

Ce chapitre nous a permis de définir rigoureusement ce qu'est un jeu de tests "meilleur" qu'un autre. Pour cela, nous avons défini plusieurs ordres (partiels) sur l'ensemble des jeux de tests d'un contexte donné : la finesse absolue, logique et formelle, puis la finesse asymptotique, logique et formelle. Nous avons montré que la finesse asymptotique logique est un ordre relativement satisfaisant pour l'intuition, alors que sa duale formelle propose un comportement beaucoup plus complexe. En particulier, des jeux de tests construits sur des hypothèses "très différentes" ne sont pas comparables.

Pour remédier à cette pauvreté des versions formelles de la finesse, nous avons défini la notion de puissance. Nous avons montré que cette notion, quoique plus riche, n'est guère plus satisfaisante, car elle ne vérifie pas le théorème de monotonie. Celui-ci, dans le cas de la finesse, montre que nos définitions sont cohérentes avec celles posées au chapitre 3.

Nous nous sommes enfin intéressé aux équivalences dérivées des ordres de finesse. Ces équivalences conservent l'acceptabilité. L'équiva-

lence asymptotique formelle semble la plus fructueuse, car elle donne lieu à une réciproque partielle pour cette propriété, qui est fausse dans le cas logique. Cette réciproque nous sera d'un usage fondamental pour toutes les questions de stratégie de construction et d'optimisation de jeux de tests développées aux chapitres suivants.

Chapitre 5 : Construction de jeux de tests ; outils théoriques et stratégie

Ce chapitre a pour but de présenter les outils théoriques qui nous seront nécessaires, aux chapitres suivants, pour traiter des exemples concrets. Ces outils seront utilisés dans le cadre d'une méthode, d'une stratégie dont nous faisons une description précise.

Pour plusieurs raisons, nous avons choisi de faire de la description théorique des outils et de la stratégie de construction un chapitre autonome, placé avant les exemples concrets d'application. Il paraîtra sans doute aride à de nombreux lecteurs, et pourra être laissé en première lecture. Les chapitres suivants y faisant souvent référence, le lecteur pourra alors, s'il le souhaite, se reporter aux paragraphes invoqués.

Au chapitre 3, nous avons décrit la notion d'acceptabilité d'un jeu de tests pour un contexte donné. Les exemples fournis montrent clairement qu'en général il n'existe pas de jeu de tests "naturel" acceptable sur le contexte proposé, mais seulement sur un contexte

restreint à partir de celui-ci. La seconde phase, dite phase de construction, d'un processus de test fait donc intervenir de manière cruciale une suite d'opérations de restriction du contexte.

$$\text{contexte } \mathcal{C}_0 \xrightarrow{\text{construction}} \begin{array}{l}\text{jeu de tests acceptable } \mathcal{T} \\ \text{sur un contexte } \mathcal{C}_1 \\ \text{restreint à partir de } \mathcal{C}_0\end{array}$$

Pour guider ces restrictions successives, et parallèlement progresser vers la solution, nous aurons besoin d'outils de réduction de problème, nous permettant de nous ramener à des cas plus simples ou mieux connus.

Une fois une solution trouvée, nous pourrons chercher à l'améliorer, selon des critères divers. C'est l'objet de l'optimisation, et des outils qui s'y rattachent.

Enfin, une fois ces outils présentés, nous pourrons décrire et justifier les grandes lignes de la stratégie qui sera appliquée aux chapitres 7 et 8.

5.1. Restriction de contexte.

Nous avons vu qu'étant donné un contexte de test, il n'existe pas toujours de jeu de tests acceptable sur ce contexte.

Deux causes peuvent être dégagées. Tout d'abord, le langage peut ne pas être assez étendu pour formuler les expériences nécessaires. Considérons par exemple le contexte $\mathcal{C} = \langle L, S, (\mathcal{S}), A \rangle$ avec $L = \{P\}$, $A = \{P(x_0)\}$, P n'étant pas d'interprétation calculable en général. Il nous faut alors passer par l'intermédiaire d'un autre prédicat, Q, d'interprétation calculable sur toute structure de la famille $\mathcal{S}$, et tel que $\forall x\,(Q(x) \rightarrow P(x))$. Un jeu de tests (logiquement) acceptable sera alors $T_n = \{Q(x_0)\}$, qui est bien un test ! Dans ce cas, il nous faut donc étendre le langage et les structures.

Une autre cause est la taille et la diversité de la famille de structures considérée. Supposons, par exemple, que nous considérions le contexte $\mathcal{C} = \langle L, S, (\mathcal{S}), A \rangle$, avec $L = \{P\}$, $S = \mathbb{N}$, $A = \{\forall x\, P(x)\}$, $(\mathcal{S})$ étant la famille des L-structures sur $\mathbb{N}$ telles que P soit d'interprétation calculable. Nous avons montré qu'il est nécessaire de poser une hypothèse d'uniformité (ou de régularité).

Par exemple $H = \{ P(0) \wedge \ldots \wedge P(k) \rightarrow \forall x\, P(x) \}$.
L'on peut prendre alors $T_n = \{ P(0) \wedge \ldots \wedge P(n) \}$.
Mais $\mathcal{C} = \langle H, (T_n)_{n \in \mathbb{N}} \rangle$ n'est pas un jeu de tests sur $\mathcal{C}$, car il existe des structures $\mathcal{S}$ de $(\mathcal{S})$ telles que $\mathcal{S} \not\models H$! Il faut donc restreindre $\mathcal{C}$ aux structures validant H, ce qui conduit à un contexte restreint sur lequel $\mathcal{C}$ est acceptable.

5.1.1. Définitions

Nous sommes donc conduit à la définition suivante.

<u>définition</u> : restriction de contexte

Soit $\mathcal{C} = \langle L, S, (\mathcal{S}), A \rangle$ un contexte de test,
et $\mathcal{C}' = \langle L', S', (\mathcal{S}'), A' \rangle$ un autre contexte.
$\mathcal{C}'$ est dit restriction de $\mathcal{C}$ si
L' est une <u>extension</u> de L,
$S' = S$
$(\mathcal{S}')_{|L(S)} \subseteq (\mathcal{S})$
$A' = A_{|L'(S)}$.

Rappelons que L' est une extension de L si tous les symboles non-logiques de L sont des symboles non-logiques de L' (avec les mêmes arités !). Cette notion correspond donc à l'inclusion de signatures dans le cas des types abstraits algébriques. (cf. annexe 3).

A l'inclusion $L \subseteq L'$ est associé un "foncteur d'oubli". Celui-ci associe canoniquement une $L(S)$-structure à une $L'(S)$-structure en oubliant les opérations de $L' \setminus L$; elle est notée $-|L(S)$

A l'inclusion $L \subseteq L'$ correspond aussi une application qui à toute $L(S)$-théorie associe la $L'(S)$-théorie ayant les mêmes axiomes non-logiques, considérés maintenant comme $L'(S)$-formules. Cette application est notée $-|L'(S)$. En pratique, si A est un $L(S)$-théorie, l'on peut sans dommage identifier A et $A|L'(S)$.

La relation de restriction est notée $\subseteq$.

<u>proposition</u>:

La relation de restriction est réflexive, anti-symétrique et transitive sur la famille des contextes de test.

<u>démonstration</u>:

Il suffit de remarquer que si $L_1 \subseteq L_2 \subseteq L_3$, et si $\mathcal{S}_3$ est une $L_3(S)$-structure,

$$\mathcal{S}_3|L_2(S)|L_1(S) = \mathcal{S}_3|L_1(S)$$

▨

Cette proposition souligne le fait que la famille des contextes de tests n'est pas un ensemble.

5.1.2. Restriction conservative.

Dans la description de la notion de processus de test, nous n'avons envisagé que certaines formes de restrictions, dites conservatives (cf. 2.1.2.3). Intuitivement, une restriction est conservative, si elle conserve toutes les structures $\mathcal{S}$ de $(\mathcal{S})$ modèles de la théorie sous test, A. Pour tenir compte du rôle du langage L, nous posons la définition suivante.

<u>définition</u> : restriction conservative.

Soit $\mathcal{C}_0 = \langle L_0, S, (\mathcal{S}_0), A \rangle$ un contexte de test, et soit $\mathcal{C} = \langle L, S, (\mathcal{S}), A \rangle$ une restriction de $\mathcal{C}$. $\mathcal{C}$ est dite restriction conservative de $\mathcal{C}_0$ si pour toute structure $\mathcal{S}_0$ de $(\mathcal{S}_0)$ telle que $\mathcal{S}_0 \models A$, il existe une structure $\mathcal{S}$ de $(\mathcal{S})$ telle que $\mathcal{S}/L_0(S) = \mathcal{S}_0$.

Nous noterons $\mathcal{C} \underset{\text{cons.}}{\subseteq} \mathcal{C}_0$. Remarquons que si $\mathcal{S}_0 \models A$ et si $\mathcal{S}/L_0(S) = \mathcal{S}_0$, alors $\mathcal{S} \models A_{/L(S)}$, ces structures étant construites sur le même univers S.

<u>proposition</u>

La relation de restriction conservative est réflexive, anti-symétrique et transitive sur la famille des contextes de test.

Elle entraîne celle de restriction.

<u>démonstration</u>:

Soit $\mathcal{C}_1 \underset{cons}{\subseteq} \mathcal{C}_2 \underset{cons.}{\subseteq} \mathcal{C}_3$, avec $\mathcal{C}_i = \langle L_i, S, (\mathcal{S}_i), A \rangle$, pour $i = 1,2,3$. Considérons $\mathcal{S}_3 \in (\mathcal{S}_3)$, avec $\mathcal{S}_3 \models A$. Il existe $\mathcal{S}_2 \in (\mathcal{S}_2)$ telle que $\mathcal{S}_2/L_3(S) = \mathcal{S}_3$, et par la remarque précédente, $\mathcal{S}_2 \models A$. D'où $\mathcal{S}_1$ telle que $\mathcal{S}_1/L_2(S) = \mathcal{S}_2$. Donc $\mathcal{S}_1/L_2(S)/L_3(S) = \mathcal{S}_3$. Donc $\mathcal{S}_1/L_3(S) = \mathcal{S}_3$.

▨

Cette proposition nous montre que, pratiquement, nous pourrons nous contenter de pratiquer des restrictions conservatives élémentaires au fur et à mesure des besoins de construction. Nous pourrons alors, par transitivité, en déduire une restriction conservative globale une fois le jeu de tests construit.

La proposition suivante montre que le fait de restreindre conservativement le contexte facilite effectivement la construction d'un jeu de tests acceptable. Si un jeu de tests acceptable existait sur le contexte primitif.

son transformé est acceptable sur le contexte restreint.

<u>proposition</u> :

Soit $\mathcal{C}_0 = \langle L_0, S, (\mathcal{S}_0), A \rangle$ un contexte de test, et $\mathcal{C} = \langle L, S, (\mathcal{S}), A \rangle$ une restriction conservative de $\mathcal{C}_0$.

Soit $\mathcal{T} = \langle H, (T_n)_{n \in \mathbb{N}} \rangle$ un jeu de tests sur $\mathcal{C}_0$. Alors $\mathcal{T}_{|L(S)} = \langle H_{|L(S)}, (T_{n|L(S)})_{n \in \mathbb{N}} \rangle$ est un jeu de tests sur $\mathcal{C}$.

$\mathcal{T}$ est logiquement acceptable si et seulement si $\mathcal{T}'$ l'est.

$\mathcal{T}$ est (formellement) acceptable si et seulement si $\mathcal{T}'$ l'est.

<u>démonstration</u> :

Montrons que $\mathcal{T}_{|L(S)}$ est un jeu de tests sur $\mathcal{C}$. Les propriétés formelles se conservant, il suffit de montrer que $T_{n|L(S)}$ est un test sur $(\mathcal{S})$, et que pour toute structure $\mathcal{S}$ de $(\mathcal{S})$, $\mathcal{S} \models H_{|L(S)}$. Puisque $(\mathcal{S})_{|L_0(S)} \subseteq (\mathcal{S})$, l'interprétation des symboles apparaissant dans les expériences de $T_{n|L(S)}$ est calculable sur toute structure de $(\mathcal{S})$. De même, puisque $(\mathcal{S})_{L_0(S)} \subseteq (\mathcal{S})$, $\mathcal{S}_{|L_0(S)} \models H$, et donc $\mathcal{S} \models H_{|L(S)}$.

Supposons $\mathcal{T}$ logiquement acceptable. Soit $\mathcal{S} \in (\mathcal{S})$, telle que $\mathcal{S} \models T_{n|L(S)}$ $\forall n \in \mathbb{N}$. Alors $\mathcal{S}_{|L_0(S)} \models T_n$ $\forall n \in \mathbb{N}$ et $\mathcal{S}_{|L_0(S)} \models A$. Donc $\mathcal{S} \models A_{|L(S)}$

Réciproquement, si $\mathcal{S} \models A_{|L(s)}$ alors $\mathcal{S}_{|L_0(s)} \models A$, et $\mathcal{S}_{|L_0(s)} \models T_n$ $\forall n \in \mathbb{N}$. Donc $\mathcal{S} \models T_{n|L(s)}$ $\forall n \in \mathbb{N}$.

Supposons maintenant $\mathcal{T}_{|L(s)}$ logiquement acceptable. Soit $\mathcal{S}_0 \in (\mathcal{S}_0)$ telle que $\mathcal{S}_0 \models T_n$ $\forall n \in \mathbb{N}$. Soit $\mathcal{S} \in (\mathcal{S})$ telle que $\mathcal{S}_{|L_0(s)} = \mathcal{S}_0$. Alors $\mathcal{S}_{|L_0(s)} \models T_n$ $\forall n \in \mathbb{N}$. Donc $\mathcal{S} \models T_{n|L(s)}$ $\forall n \in \mathbb{N}$. Donc $\mathcal{S} \models A$ et $\mathcal{S}_0 \models A$. Réciproquement, si $\mathcal{S}_0 \models A$, $\mathcal{S} \models A$, $\mathcal{S} \models T_{n|L(s)}$, $\mathcal{S}_{|L_0(s)} \models T_n$ et $\mathcal{S}_0 \models T_n$, et ce $\forall n \in \mathbb{N}$.

faux

Dans le cas formel, si $\mathcal{T}$ est acceptable, alors $\bigsqcup_{n \in \mathbb{N}} T_n \sqcup H \vdash A \sqcup H$. Donc $\bigsqcup_{n \in \mathbb{N}} T_{n|L(s)} \sqcup H_{|L(s)} \vdash A_{|L(s)} \sqcup H_{|L(s)}$, et $\mathcal{T}_{|L(s)}$ l'est.

Si $\mathcal{T}_{|L(s)}$ est acceptable,

$\bigsqcup_{n \in \mathbb{N}} T_{n|L(s)} \sqcup H_{|L(s)} \vdash A_{|L(s)} \sqcup H_{|L(s)}$.

Donc $\bigsqcup_{n \in \mathbb{N}} T_n \sqcup H \vdash A \sqcup H$, mais les déduction peuvent éventuellement faire intervenir des symboles de $L \setminus L_0$! Le corollaire au théorème de complitude (cf. annexe 3) montre que l'on peut trouver des déductions ne faisant intervenir que des symboles de L_0, ce qui montre que $\mathcal{T}$ est acceptable

▨

5.1.3. Cas particuliers

Nous examinons maintenant quelques cas particuliers de restrictions conservatives qui nous seront utiles par la suite. Ils correspondent aux problèmes évoqués dans l'introduction à cette partie.

Soit $\mathcal{C}_0 = \langle L_0, S, (\mathcal{S}_0), A \rangle$ un H_0-contexte. Nous cherchons à construire un jeu de tests de la forme $\langle H_0 \sqcup H_1, (T_n)_{n \in \mathbb{N}} \rangle$. Il faut donc restreindre $\mathcal{C}_0$ à un $H_0 \sqcup H_1$-contexte. La proposition suivante montre que cela est possible sous la condition (suffisante mais non nécessaire)

$$H_0 \sqcup A \vdash H_1$$

<u>proposition</u> :

Soit $\mathcal{C}_0 = \langle L_0, S, (\mathcal{S}_0), A \rangle$ un H_0-contexte, et soit H_1 telle que $H_0 \sqcup A \vdash H_1$

Alors $\mathcal{C} = \langle L_0, S, (\mathcal{S}), A \rangle$, avec

$$(\mathcal{S}) = \{ \mathcal{S}_0 \in (\mathcal{S}_0),\ \mathcal{S}_0 \models H_1 \}$$

est une restriction conservative de $\mathcal{C}_0$, et est un $H_0 \sqcup H_1$-contexte.

<u>démonstration</u> :

Par le théorème de validité (cf. annexe 3).

On a, pour toute structure $\mathcal{S}_0 \in (\mathcal{S}_0)$, $\mathcal{S}_0 \models H_0$; si $\mathcal{S}_0 \models A$, alors $\mathcal{S}_0 \models H_0 \sqcup A$, et $\mathcal{S}_0 \models H_1$; donc $\mathcal{S}_0 \in (\mathcal{S})$.

▨

En particulier, en prenant $H_1 = A$, le contexte obtenu est un A-contexte. Le jeu de tests $\langle A, (\emptyset)_n \rangle$ y est donc acceptable. Nous pouvons donc énoncer la proposition suivante.

<u>proposition</u> :

Soit $\mathcal{C}$ un jeu de tests.

Il existe une restriction conservative de $\mathcal{C}$ admettant un jeu de tests acceptable.

Il est bien évident que cette proposition n'a pas d'intérêt pratique, car la famille de structures du contexte obtenue est en général trop "petite", trop "clairsemée", pour que le <u>principe de couplage</u> soit applicable (cf. 2.1.4.). Nous n'appliquerons donc ce type de restriction qu'avec des hypothèses H, vérifiées "<u>probablement</u>" par l'objet sous test.

Un autre cas est celui où nous étendons le langage, en restreignant éventuellement l'interprétation des nouveaux symboles par de nouvelles hypothèses H.

Soit $\mathcal{C}_0 = \langle L_0, S, (\mathcal{P}_0), A \rangle$. Soit $L \supseteq L_0$. Nous supposerons que toute structure de $(\mathcal{P}_0)$ s'étend en une L-structure modèle de H. La proposition suivante montre que la restriction

de contexte obtenue est bien conservative.

<u>proposition</u> :

Soit $\mathcal{C}_0 = \langle L_0, S, (\mathcal{S}_0), A \rangle$ un H_0-contexte, et L une extension de L_0.

Soit H une $L(S)$-théorie telle que toute structure $\mathcal{S}_0$ de $(\mathcal{S}_0)$ s'étende en une $L(S)$-structure modèle de H.

Alors le contexte $\mathcal{C} = \langle L, S, (\mathcal{S}), A \rangle$, avec $(\mathcal{S}) = \{\mathcal{S}, L(S)\text{-structure}, \mathcal{S} \models H, \mathcal{S}_{|L_0(S)} \in (\mathcal{S}_0)\}$ est une restriction conservative de $\mathcal{C}_0$, et est un $H \sqcup H_0$-contexte.

<u>démonstration</u> :

Par construction, c'est un $H \sqcup H_0$-contexte.

D'autre part, par hypothèse, pour toute structure $\mathcal{S}_0$ de $(\mathcal{S}_0)$, il existe une structure $\mathcal{S}$ de $(\mathcal{S})$ telle que $\mathcal{S}_{|L_0(S)} = \mathcal{S}_0$

▨

Un cas particulier est celui où tous les axiomes non-logiques de H sont des $(L \setminus L_0)(S)$ formules. La condition de la proposition se réduit au fait que H admette un modèle sur S.

Un autre cas est bien sûr celui où H est vide !

5.1.4. Conclusion.

Cette partie nous a permis de développer la notion de restriction de contexte, et plus spécialement de restriction de contexte conservative.

Nous avons montré la transitivité de telles restrictions, ce qui nous autorise à progresser par restrictions élémentaires.

Nous avons montré que de telles restrictions ne nous font pas "perdre" de jeux de tests acceptables, mais ne peuvent qu'en créer. De plus, étant donné un contexte, il existe toujours un contexte conservativement restreint admettant un jeu de tests acceptable (quoique trivial).

Enfin, nous avons montré que, sous certaines conditions, le fait d'ajouter des hypothèses ou d'étendre le langage conduit <u>automatiquement</u> à des restrictions conservatives. Cependant, l'<u>opportunité</u> et la <u>rentabilité</u> de telles restrictions devront, dans la pratique, être évaluées avec soin, afin de rester dans le champ d'application du <u>principe de couplage</u>.

5.2. Réduction de problème

La partie précédente a montré comment, par restrictions successives du contexte, l'on peut réunir les conditions d'existence d'un jeu de tests acceptable.

Cependant, cette méthode cause une perte de qualité du processus de test, puisque, la famille des structures du contexte étant de plus en plus réduite, le principe du couplage se justifie de moins en moins.

Aussi développons-nous dans cette partie des méthodes permettant de progresser vers une solution, de réduire le problème sans perte de qualité.

5.2.1. Théorème de localisation

Ce théorème s'intéresse au cas où la théorie sous test, A, est sous la forme d'une réunion <u>dénombrable</u> de sous-théories : $A = \bigsqcup_{i \in \mathbb{N}} A^{(i)}$.
Le théorème montre alors que pour construire un jeu de tests acceptable pour A, il suffit d'en construire un pour chacun des $A^{(i)}$ sur un contexte adéquat, ce qui est, en général, plus aisé.

<u>Théorème de localisation</u> (cas général)

Soit $\mathcal{C} = \langle L, S, (\mathcal{S}), A \rangle$ un contexte de test, A étant de la forme $\bigsqcup_{i \in \mathbb{N}} A^{(i)}$, où les $A^{(i)}$ sont des $L(S)$-théories.

Pour chaque indice i, soit $\mathcal{C}^{(i)}$ le contexte $\langle L, S, (\mathcal{S}), A^{(i)} \rangle$, et soit $\mathcal{T}^{(i)} = \langle H^{(i)}, (T_n^{(i)})_{n \in \mathbb{N}} \rangle$ un jeu de tests sur $\mathcal{C}^{(i)}$.

Posons $H = \bigsqcup_{i \in \mathbb{N}} H^{(i)}$

$$T_n = \bigsqcup_{i \leq n} T_n^{(i)} = T_n^{(0)} \sqcup \dots \sqcup T_n^{(n)}$$

Alors $\mathcal{T} = \langle H, (T_n)_{n \in \mathbb{N}} \rangle$ est un jeu de tests sur $\mathcal{C}$.

Si chaque $\mathcal{C}^{(i)}$ est logiquement acceptable, $\mathcal{C}$ l'est.

Si chaque $\mathcal{C}^{(i)}$ est formellement acceptable, $\mathcal{C}$ l'est.

Pour illustrer cet énoncé barbare, prenons $S = \mathbb{N}$, $L = \{\mathcal{I}\}$, $A = \{\mathcal{I}(0), \mathcal{I}(1), \dots\}$, $(\mathcal{S})$ étant la famille des $L(S)$-structures telles que $\mathcal{I}$ soit d'interprétation calculable.

Prenons $A^{(i)} = \{\mathcal{I}(i)\}$. $H^{(i)} = \emptyset$ et $T_n^{(i)} = \{\mathcal{I}(i)\}$ conviennent. $\mathcal{C}^{(i)}$ est alors formellement acceptable.

Nous avons donc $H = \emptyset$, $T_n = \{\mathcal{I}(0), \dots, \mathcal{I}(n)\}$ et $\mathcal{T} = \langle H, (T_n)_{n \in \mathbb{N}} \rangle$ est formellement acceptable.

Mais T_n est formellement équivalent à $T'_n = \{ P(0) \wedge \dots \wedge P(n) \}$, et nous retrouvons donc un jeu bien connu des lecteurs attentifs...

Passons à la démonstration du théorème.

<u>démonstration</u> :

① Il faut d'abord vérifier que $\mathcal{C}$ est bien un jeu de tests sur $\mathcal{C}$.

ⓐ Fixons n. Les axiomes non-logiques de $T_n^{(0)}, \dots, T_n^{(n)}$ sont des $(\mathcal{S})$-expériences, en nombre fini. Comme l'on ne considère qu'un réunion finie, ceux de T_n aussi, et T_n est donc un test sur $\mathcal{C}$.

ⓑ Soit $\mathcal{S} \in (\mathcal{S})$. $\mathcal{S} \models H^{(i)}$ $\forall i \in \mathbb{N}$ par hypothèse. Donc $\mathcal{S} \models \bigsqcup_{i \in \mathbb{N}} H^{(i)}$, et $\mathcal{S} \models H$.

ⓒ Fixons n, et étudions $T_{n+1} \sqcup H$. Cela se réécrit $\left(T_{n+1}^{(0)} \sqcup \dots \sqcup T_{n+1}^{(n+1)}\right) \sqcup \left(H^{(0)} \sqcup H^{(1)} \sqcup \dots\right)$.
En regroupant, nous pouvons le réécrire

$$\left[T_{n+1}^{(0)} \sqcup H^{(0)}\right] \sqcup \left[\dots \qquad \dots\right] \sqcup \left[T_{n+1}^{(n+1)} \sqcup H^{(n+1)}\right] \sqcup \bigsqcup_{i \geqslant n+2} H^{(i)} \qquad (1)$$

Mais, par hypothèse, puisque $\mathcal{C}^{(i)}$ est un jeu de tests,

$$T_{n+1}^{(i)} \sqcup H^{(i)} \vdash T_n^{(i)} .$$

Donc (1) démontre

$$T_n^{(0)} \sqcup \dots \sqcup T_n^{(n)} \sqcup T_n^{(n+1)} \sqcup \bigsqcup_{i \geqslant n+2} H^{(i)}$$

et donc, en particulier

$$T_n^{(0)} \sqcup \dots \sqcup T_n^{(n)}, \text{ soit } T_n .$$

② Supposons les $\mathcal{C}^{(i)}$ formellement acceptables.

ⓐ Montrons que, les $\mathcal{C}^{(i)}$ étant formellement asymptotiquement valides, $\mathcal{C}$ l'est.

$\bigsqcup_n T_n \sqcup H$ se réécrit

$\bigsqcup_n (T_n^{(0)} \sqcup \ldots \sqcup T_n^{(n)}) \bigsqcup_i H^{(i)}$, soit

$$\bigsqcup_i \left(\bigsqcup_{n \geq i} T_n^{(i)} \sqcup H^{(i)} \right) \qquad (2)$$

Mais $\mathcal{C}^{(i)}$ étant formellement projectivement fiable, $T_i^{(i)} \sqcup H^{(i)} \vdash \bigsqcup_{n \leq i} T_n^{(i)} \sqcup H^{(i)}$

Donc (2) démontre

$$\bigsqcup_i \left(\bigsqcup_n T_n^{(i)} \sqcup H^{(i)} \right),$$

et par hypothèse de validité, cela démontre $\bigsqcup_i A^{(i)}$, soit A.

ⓑ Montrons que, les $\mathcal{C}^{(i)}$ étant formellement non-biaisés, $\mathcal{C}$ l'est.

$A \sqcup H$ s'écrit $\bigsqcup_i (A^{(i)} \sqcup H^{(i)})$, ce qui démontre $\bigsqcup_i (T_n^{(i)})$ pour tout n, et donc, en particulier $\bigsqcup_{i \leq n} T_n^{(i)}$, soit T_n.

③ Supposons les $\mathcal{C}^{(i)}$ logiquement acceptables.

ⓐ Montrons que les $\mathcal{C}^{(i)}$ étant logiquement asymptotiquement valides, $\mathcal{C}$ l'est.

Soit $\mathcal{L} \in (\mathcal{L})$, $\mathcal{L} \models T_n$ $\forall n \in \mathbb{N}$. Fixons n.

$\mathcal{L} \models T_n$, donc $\mathcal{L} \models T_n^{(0)} \sqcup \ldots \sqcup T_n^{(n)}$.

Donc $\mathcal{L} \models T_n^{(i)}$ $\forall i \leq n$, et ce $\forall n \in \mathbb{N}$.

Mais par fabilité projective logique des $\mathcal{T}^{(i)}$, ceci montre que $\mathcal{S} \models T_n^{(i)}$ $\forall i \in \mathbb{N}$ $\forall n \in \mathbb{N}$.
Fixons alors i. $\mathcal{S} \models T_n^{(i)}$ $\forall n$, donc, par hypothèse de validité, $\mathcal{S} \models A^{(i)}$, et ce $\forall i \in \mathbb{N}$.
Donc $\mathcal{S} \models \bigsqcup_{i \in \mathbb{N}} A^{(i)}$, et $\mathcal{S} \models A$.

ⓑ Montrons que les $\mathcal{T}^{(i)}$ étant logiquement non-biaisés, $\mathcal{T}$ l'est.
Soit $\mathcal{S} \in (\mathcal{S})$, $\mathcal{S} \models A$. Alors $\mathcal{S} \models \bigsqcup_{i \in \mathbb{N}} A^{(i)}$.
Donc $\mathcal{S} \models A^{(i)}$ $\forall i \in \mathbb{N}$.
Fixons i. Par l'hypothèse d'absence de biais pour $\mathcal{T}^{(i)}$, $\mathcal{S} \models T_n^{(i)}$ $\forall n \in \mathbb{N}$, et ce $\forall i \in \mathbb{N}$.
Donc $\mathcal{S} \models T_n^{(0)} \sqcup \ldots \sqcup T_n^{(n)}$, soit T_n, et ce $\forall n \in \mathbb{N}$.

▨

5.2.2. Discussion.

Le théorème de localisation n'a, en général, pas d'équivalent dans le cas d'une famille $A^{(i)}$ non-dénombrable. Le procédé de diagonalisation utilisé n'est alors plus valable ; d'autre part, nous avons imposé <u>arbitrairement</u> aux familles de tests des jeux de tests d'être dénombrables.

Dans la pratique, ce théorème nous permet de ramener la construction d'un jeu de tests pour une famille de formules à celle d'un jeu de tests pour <u>une seule formule</u>. Si nous sommes assuré qu'il n'y a qu'une infinité dénombrable de formules distinctes, le théorème sera donc toujours applicable. Par contre, s'il est possible de devoir tester conjointement une infinité non-dénombrable de formules, alors nous ne sommes plus assurés que le théorème suffira toujours à nous ramener au cas d'une formule.

Une condition <u>suffisante</u> pour garantir que le théorème soit toujours efficace est de se restreindre à un langage L dénombrable (n'ayant qu'une quantité dénombrable de symboles non-logiques), et à un ensemble S dénombrable. L'ensemble des L(S)-formules est alors dénombrable.

En un certain sens, donc, le cas $S = \mathbb{N}$ est le seul cas pour lequel nous disposons d'outils effectifs de construction de jeux de tests acceptables. Ceci justifie, a posteriori, que tous nos exemples soient de cette forme.

Remarquons que le formalisme des types abstraits algébriques se restreint, lui aussi,

en pratique à ce cas. Les signatures Σ sont finies, ou, plus rarement, dénombrables. Les seules algèbres étudiées sont des quotients de l'algèbre des termes, qui est dénombrable dans ce cas [ADJ 78], [Gogu 77], [CIR 80] etc.
Nous restons donc, malgré ces nouvelles restrictions, dans le cadre de ce formalisme (cf. chapitre 7)

5.2.3. Propriétés connexes.

Dans le cas où l'on considère une famille finie, et non infinie, $A^{(i)}$, le théorème de localisation peut se mettre sous une forme simplifiée.

<u>Théorème de localisation</u> (cas fini)

Soit $\mathcal{C} = \langle L, S, (\mathcal{S}), A \rangle$ un contexte de test, A étant de la forme $A^{(0)} \sqcup \dots \sqcup A^{(k)}$, où les $A^{(i)}$ sont des $L(S)$-théories.

Pour chaque i entre 0 et k, soit $\mathcal{C}^{(i)}$ le contexte $\langle L, S, (\mathcal{S}), A^{(i)} \rangle$, et soit $\mathcal{T}^{(i)} = \langle H^{(i)}, (T_n^{(i)})_{n \in \mathbb{N}} \rangle$ un jeu de tests sur $\mathcal{C}^{(i)}$.

Alors $\mathcal{T} = \langle H, (T_n)_{n \in \mathbb{N}} \rangle$ est un jeu de tests sur $\mathcal{C}$, avec

$$H = H^{(0)} \sqcup \dots \sqcup H^{(k)}$$
$$T_n = T_n^{(0)} \sqcup \dots \sqcup T_n^{(k)}$$

Si chaque $\mathcal{T}^{(i)}$ est logiquement acceptable, $\mathcal{T}$ l'est.

Si chaque $\mathcal{T}^{(i)}$ est formellement acceptable, $\mathcal{T}$ l'est.

Le théorème de localisation est l'outil théorique dont nous disposons pour construire un jeu de tests acceptable sur un contexte donné. Cependant A ne se présente pas toujours sous la forme $\bigsqcup_{i \in \mathbb{N}} A^{(i)}$. Il faut donc pouvoir se ramener à ce cas en modifiant, sous certaine conditions, le contexte de test.

<u>proposition</u> :

Soit $\mathcal{C}_1 = \langle L, S, (\vartheta), A_1 \rangle$ et $\mathcal{C}_2 = \langle L, S, (\vartheta), A_2 \rangle$ deux H_0-contextes, et soit $\mathcal{T} = \langle H, (T_n)_{n \in \mathbb{N}} \rangle$ un jeu de tests sur $\mathcal{C}_1$ (et donc sur $\mathcal{C}_2$!)

Supposons $H \vdash H_0$, et $H_0 \sqcup A_1 \vdash H_0 \sqcup A_2$ (H_0-équivalence formelle).

Alors $\mathcal{T}$ est formellement acceptable pour $\mathcal{C}_1$ si et seulement si il l'est pour $\mathcal{C}_2$.

<u>démonstration</u> :

Supposons, par symétrie, que $\mathcal{T}$ est formellement acceptable pour $\mathcal{C}_1$.

Examinons $H \sqcup A_2$. Cela démontre $H \sqcup H_0 \sqcup A_2$, donc $H \sqcup H_0 \sqcup A_1$, donc $H \sqcup A_1$, donc $H \bigsqcup_{n \in \mathbb{N}} T_n$.

Réciproquement, $H \cup \bigcup_{n \in \mathbb{N}} T_n$ démontre $H \cup A_1$, et donc $H \cup H_0 \cup A_1$, $H \cup H_0 \cup A_2$, soit $H \cup A_2$

▨

Donc, pour se ramener aux hypothèses du théorème de localisation, on peut transformer A par H_0-équivalence formelle, si, par la suite, l'on se restreint aux jeux de tests dont les hypothèses recouvrent H_0.

Soit, par exemple, $\mathcal{C}_0 = \langle L, S, (\mathcal{I}_0), A \rangle$, avec $S = \mathbb{N}$, $L = \{P\}$, $A = \{\forall x\, P(x)\}$, $(\mathcal{I}_0)$ étant la famille de toutes les $L(S)$-structures telles que P soit d'interprétation calculable. $\mathcal{C}$ n'admet pas, en général de jeu de tests acceptable.

Posons $H_1 = \{P(0) \wedge \ldots \wedge P(k) \rightarrow \forall x\, P(x)\}$, et appliquons 5.1.3. Nous obtenons un H_1-contexte, $\mathcal{C}$, restriction conservative de $\mathcal{C}_0$.

Appliquons la proposition précédente. Sous H_1, A est equivalente à $A' = \{P(0), P(1), \ldots\}$.

$\mathcal{C}' = \langle L, S, (\mathcal{I}), A' \rangle$ admet, par le théorème de localisation, un jeu de tests acceptable $\langle \emptyset, (T_n)_{n \in \mathbb{N}} \rangle$, avec $T_n = \{P(0) \wedge \ldots \wedge P(n)\}$.

Donc $\mathcal{C} = \langle H_1, (T_n)_{n \in \mathbb{N}} \rangle$ est un jeu de tests acceptable de $\mathcal{C}'$, et donc de $\mathcal{C}$.

Les lecteurs attentifs auront reconnu un air déjà souvent entendu...

La proposition admet un dual logique.

<u>proposition</u> :

Soit $\mathcal{C}_1 = \langle L, S, (\mathcal{S}), A_1 \rangle$ un contexte de test, et $\mathcal{C}_2 = \langle L, S, (\mathcal{S}), A_2 \rangle$ un autre contexte.

Supposons que pour toute structure $\mathcal{S}$ de $(\mathcal{S})$, $\mathcal{S} \models A_1$ si et seulement si $\mathcal{S} \models A_2$.

Soit $\mathcal{T} = \langle H, (T_n)_{n \in \mathbb{N}^*} \rangle$ un jeu de tests sur $\mathcal{C}_1$ (et $\mathcal{C}_2$).

Alors $\mathcal{T}$ est logiquement acceptable pour $\mathcal{C}_1$, si et seulement si il l'est pour $\mathcal{C}_2$.

5.2.4. Conclusion.

Dans cette partie, nous avons présenté et étudié l'outil le plus puissant dont nous disposons pour construire effectivement un jeu de tests acceptable sur un contexte donné. Cet outil est le théorème de localisation, permettant de se ramener, à partir d'une théorie sous test complexe, au cas où celle-ci n'a qu'un seul axiome non-logique.

Ce théorème n'est applicable efficacement que si l'on est en présence d'une quantité au plus dénombrable de formules. Une condition <u>suffisante</u> pratique sera de se restreindre au cas où L et S sont dénombrables. Il n'existe alors (à α-conversion près) qu'une quantité dénombrable de formules. D'autre part, le passage de $\{\forall x\, \phi(x)\}$ à $\{\phi(s), s \in S\}$ ne donne naissance qu'à une quantité dénombrable de formules si et seulement si S est dénombrable (élimination des quantificateurs par énumération).

Nous avons proposé quelques outils permettant, sous certaines conditions, de transformer la théorie sous test par équivalence pour se ramener à un cas d'application du théorème de localisation. Ce mouvement de transformation par ajout d'hypothèses (et restriction) et localisation est à la base de notre stratégie.

5.3. Optimisation

Les parties précédentes nous ont montré comment construire un jeu de tests sur un contexte donné, en employant éventuellement des restrictions conservatives.

Nous avons donc réalisé la flèche suivante.

$$\begin{array}{c} H_0\text{-contexte initial.} \\ \mathcal{C}_0 \end{array} \xrightarrow{\text{construction}} \begin{array}{c} \text{jeu de tests } \mathcal{T}_1 \text{ sur} \\ \text{le } H_0 \cup H_1\text{-contexte} \\ \mathcal{C}_1 \text{, restriction} \\ \text{conservative de } \mathcal{C}_0 \end{array}$$

Il est possible, pour de multiples raison que $\mathcal{T}_1$ ou $\mathcal{C}_1$ ne soient pas satisfaisants. Aussi nous allons être amené à améliorer, à "optimiser" notre construction par l'introduction éventuelle de nouvelles hypothèses H_2, d'un nouveau contexte $\mathcal{C}_2$, et la construction d'un nouveau jeu de tests $\mathcal{T}_2$.

Nous considérons donc le problème suivant.

$$\begin{array}{c} \text{jeu de tests } \mathcal{T}_1 \text{ sur} \\ \text{le } H_0 \cup H_1 \text{ contexte } \mathcal{C}_1 \\ \text{restriction conservative} \\ \text{de } \mathcal{C}_0 \end{array} \xrightarrow{\text{optimisation}} \begin{array}{c} \text{jeu de tests } \mathcal{T}_2 \text{ sur} \\ \text{le } H_0 \cup H_1 \cup H_2 \text{ contexte } \mathcal{C}_2 \\ \text{restriction conservative} \\ \text{de } \mathcal{C}_1 \end{array}$$

Par transitivité de la restriction conservative, $\mathcal{C}_2$ est restriction conservative de $\mathcal{C}_0$, et $\mathcal{T}_2$ est bien aussi une solution du problème de construction.

Nous allons donc examiner différents critères d'amélioration possibles.

5.3.1. Optimisation selon l'ordre asymptotique

Cherchons d'abord, le contexte étant fixé, à améliorer notre jeu de tests selon l'ordre de finesse asymptotique formelle (cf. 4.2.1). Le corollaire au théorème d'équivalence, ou théorème fondamental du test (cf. 4.4.3.) nous montre que l'on ne peut procéder que par <u>équivalence asymptotique formelle</u>.

Soit $\mathcal{C} = \langle L, S, (P), A \rangle$ le contexte envisagé, et $\mathcal{T} = \langle H, (T_n)_{n \in \mathbb{N}} \rangle$ le jeu de tests construit. Par le théorème de stabilité (cf. 4.4.2.), <u>tout</u> jeu de test $\mathcal{T}'$ formellement asymptotiquement équivalent à $\mathcal{T}$ sera acceptable (si toutefois A ne contient qu'un nombre fini d'axiomes non logiques)

Tout d'abord, nous sommes donc autorisé à éliminer, ou, au contraire, à rajouter des expériences dans chaque test de $\mathcal{T}$, obtenant ainsi $\mathcal{T}' = \langle H, (T'_n)_{n \in \mathbb{N}} \rangle$, pourvu que

$$H \sqcup T_n \vDash H \sqcup T'_n \quad \text{(equivalence absolue formelle)}$$

En particulier, si T_n contient des expériences dont les résultats sont impliqués par H, ou par d'autres expériences, nous pourrons les supprimer. Nous pourrons simplifier la formulation des expériences, séparer les conjonctions ou au contraire les regrouper. Nous pourrons remplacer une expérience par une autre expérience H-équivalente

formellement.

D'autre part, d'après la proposition énoncée en 4.1.1, nous pouvons remplacer $\mathcal{T}$ par un test strictement extrait de $\mathcal{T}$. Nous pouvons en particulier "étirer" les indices ($T'_n = T_{E[\frac{n}{2}]}$) ou au contraire les "contracter" ($T'_n = T_{2n}$). La notion d'indice n'a donc aucune valeur intrinsèque quant à la qualité d'un test pour un problème donné. Des critères subjectifs ou extérieurs devront donc être utilisés pour le choix du k lors de la phase d'application du processus de test (cf. 2.1.3.).

Toutes ces optimisations relèvent donc de l'équivalence asymptotique formelle, et, d'un point de vue théorique, sont donc rigoureusement sans intérêt. Il en est bien sûr tout autrement d'un point de vue pratique, lorsque des critères de rentabilité, de lisibilité, de probabilité sont pris en compte. En particulier, il est évident pratiquement qu'il est souhaitable pour un test d'être "légèrement" redondant, pour éviter à des expériences d'être réussies par coïncidence ("coincidental correctness", cf. [Budd 91]).

5.3.2. Autres critères.

Il arrive souvent que l'on considère trop d'hypothèses par rapport à celles véritablement utilisées dans la construction. Il est alors possible d'éliminer les hypothèses inutiles.

proposition :

Soit $\mathcal{C} = \langle L, S, (\mathcal{S}), A \rangle$ un contexte, $\mathcal{T} = \langle H, (T_n)_{n \in \mathbb{N}} \rangle$ un jeu de tests sur ce contexte.

Soit H' une $L(S)$-théorie, avec $H \vdash H'$, telle que $H' \sqcup \bigsqcup_{n \in \mathbb{N}} T_n \models H' \sqcup A$.

Alors $\mathcal{T}' = \langle H', (T_n)_{n \in \mathbb{N}} \rangle$ est un jeu de tests acceptable sur $\mathcal{C}$.

Il est possible aussi que l'on ait trop restreint la famille de structures envisagée. Il est possible de l'augmenter, tout en gardant une extension conservative du contexte initial.

proposition :

Soit $\mathcal{C}_0$ un contexte, $\mathcal{C}_1 = \langle L, S, (\mathcal{S}_1), A \rangle$ une restriction conservative de $\mathcal{C}_0$, et $\mathcal{T} = \langle H, (T_n)_{n \in \mathbb{N}} \rangle$ un jeu de tests acceptable sur $\mathcal{C}_1$.

Soit $(\mathcal{S}_2)$ une famille de $L(S)$-structures telle que $(\mathcal{S}_1) \subseteq (\mathcal{S}_2)$, et que toute structure de $(\mathcal{S}_2)$ soit modèle de H, et $(\mathcal{S}_2)_{|L_0(S)} = (\mathcal{S}_1)_{L_0(S)}$

Alors $\mathcal{C}_2 = \langle L, S, (\mathcal{S}_2), A \rangle$ est un contexte

de test, restriction conservative de $\mathcal{C}_0$, et $\mathcal{C}$ est un jeu de tests acceptable sur $\mathcal{C}_2$.

En modifiant la théorie sous test, nous pourrons aussi tester une partie calculable des hypothèses, et donc améliorer le processus de test.

<u>proposition</u> :

Soit $\mathcal{C} = \langle L, S, (\mathcal{J}), A \rangle$ un contexte de test, et $\mathcal{T} = \langle H, (T_n)_{n \in \mathbb{N}} \rangle$ un jeu de tests sur $\mathcal{C}$, tel que H soit de la forme $H_1 \sqcup T$, où T est un test sur $\mathcal{C}$.

Alors $\mathcal{T}_1 = \langle H_1, (T_n \sqcup T)_{n \in \mathbb{N}} \rangle$ est un jeu de test sur $\mathcal{C}_1 = \langle L, S, (\mathcal{J}), A \sqcup T \rangle$.

Si $\mathcal{T}$ est acceptable sur $\mathcal{C}$, $\mathcal{T}_1$ est acceptable sur $\mathcal{C}_1$.

Enfin, dans certains cas, nous serons amené, quoiqu'ayant trouvé un jeu de test acceptable, à restreindre de nouveau le contexte par l'ajout d'hypothèses (hypothèses d'optimisation H_2) pour obtenir, par les méthodes ci-dessus, un jeu de test plus réduit. La "rentabilité" de cette opération devra être étudiée avec soin.

5.4. Stratégie de construction

Nous pouvons maintenant regrouper tous ces outils de manière cohérente pour tracer les grandes lignes de la stratégie de construction qui sera appliquée aux chapitres 6 et 7.

Nous voulons réaliser la flèche suivante.

H_0 contexte initial $\mathcal{C}_0$. $\longrightarrow$ $H_0 \cup H_1$ - contexte $\mathcal{C}_1$
jeu de tests $\langle H_0 \cup H_1, (T_n)_{n \in \mathbb{N}} \rangle$
acceptable sur $\mathcal{C}_1$
$\mathcal{C}_1$ restriction conservative de $\mathcal{C}_0$.

5.4.1. Principe

Les outils de base sont les propriétés concernant la restriction conservative de contexte (cf. 5.1.2.). La transitivité nous permet de procéder par restrictions élémentaires successives.

Ces restrictions ont pour but d'augmenter les hypothèses validées par les structures (cf. 5.1.3.) de manière à pouvoir transformer A (cf. 5.2.3) pour appliquer le théorème de localisation (cf. 5.2.1.) Nous sommes ainsi ramené au point de départ, avec une théorie sous test plus simple. En général, les hypothèses supplémentaires sont des hypothèses de regularité ou d'uniformité, et nous sommes ramené à tester une formule ayant un quantificateur (universel) de moins que celle dont nous sommes parti (cf. 5.2.3.)

En fait, ces deux procédés sont parallèles, et n'interfèrent pas. Le découplage est assuré par les propriétés exposées au paragraphe 5.1.3. Le premier fait évoluer $(\mathcal{S})$ et L (partie logique), le second H (partie formelle). Les propriétés sus-citées montrent qu'à toute manipulation élémentaire des hypothèses, l'on peut associer une transformation élémentaire de $(\mathcal{S})$ et L, du moins sous certaines conditions.

Les propriétés développées au paragraphe 5.3.2. montrent qu'il suffit de se préoccuper de la partie formelle, rajoutant des axiomes et des symboles selon les besoins. Il est possible, un jeu de tests étant construit, d'éliminer les hypothèses superflues, et rajouter des structures, si la famille a été implicitement trop restreinte.

La phase de construction est donc, malgré les apparences, <u>purement formelle</u>, tout le problème se résumant à trouver des théories H et T_n telles que $H \cup A \models H \cup \bigcup_{n \in \mathbb{N}} T_n$. Le seul problème est d'être assuré au terme de cette phase que le principe de couplage reste valable; en d'autre termes, que la structure représentant réellement l'objet sous test "n'est pas trop loin" de l'ensemble

des structures du contexte étudié. Il suffit pour cela, vu la méthode, de n'avoir utilisé que des hypothèses probablement vérifiées par l'objet sous test.

5.4.2. Description

La stratégie se décompose donc ainsi.

5.4.2.1 De l'état initial

Le contexte initial est construit à partir des données du problème, soit $\mathcal{C}_0 = \langle L_0, S, (\mathcal{P}_0), A \rangle$. Nous construisons H_0, constituant les <u>hypothèses initiales</u>, telle que $\mathcal{C}_0$ soit un H_0-contexte. H_0 peut être la plus catégorique, la plus précise possible ; ces hypothèses sont considérées comme "très probablement vérifiées."

5.4.2.2 De la construction

Nous construisons un jeu de tests acceptable $\mathcal{T}^1$ de la forme $\langle H_0 \cup H_1, (T_n^1)_{n \in \mathbb{N}} \rangle$ sur un contexte virtuel dérivé de $\mathcal{C}_0$, $\mathcal{C}_1' = \langle L_1, S, \emptyset, A \rangle$, avec $L_1 \supseteq L_0$.

En d'autres termes, nous construisons, éventuellement en utilisant de nouveaux symbo-

les, des théories H_1 et T_n^1, $n \in \mathbb{N}$, telles que $H_1 \cup H_0 \cup A \vdash \bigsqcup_{n \in \mathbb{N}} T_n^1 \cup A$, et ceci sans nous préoccuper de $(\mathcal{S}_0)$.

Nous nous contenterons de deux types d'hypothèses :

- des hypothèses de régularité ou d'uniformité, telles que $H_0 \cup A \vdash H_1$
- des hypothèses réglant le comportement des nouveaux symboles, non-contradictoires avec les structures de $(\mathcal{S}_0)$.

Comme nous l'avons déjà précisé, nous ne considérons que des problèmes tels que

- L_0 soit un langage dénombrable,
- S soit dénombrable,
- $(\mathcal{S})$ soit telle que tous les symboles de L soient d'interprétation calculable dans chacune des structures,
- A soit constitué d'axiomes non-logiques purement universels.

Il y a au plus une quantité dénombrables de formules dans ce cas, et l'on peut supposer A de la forme $\{\forall x\ \phi(x)\}$ par localisation

Si, <u>intuitivement</u>, l'hypothèse d'uniformité $H_1 = \{\exists x\ \phi(x) \rightarrow \forall x\ \phi(x)\}$ est "probablement vraie", comme elle vérifie $H_0 \cup A \vdash H_1$, nous l'ajoutons aux hypothèses.

Considérant alors un nouveau symbole a, nous sommes ramené à considérer le problème de la construction d'un jeu de tests pour

$A = \{\phi(a)\}$.

Si ϕ ne contient pas de variable,

$T_n = \{\phi(a)\}$ $\forall n \in \mathbb{N}$ convient, par l'hypothèse de calculabilité des symboles de L.

Sinon, nous recommençons le processus.

Dans d'autres cas, il sera plus <u>intuitif</u> de poser une hypothèse de la forme

$H_1 = \{\phi(s_{i_1}) \wedge \dots \wedge \phi(s_{i_k}) \rightarrow \forall x\, \phi(x)\}$, qui vérifie bien $H_0 \sqcup A \vdash H_1$, avec $(s_i)_{i \in \mathbb{N}}$ énumération de S.

Alors, sous H_1, la recherche d'un jeu de tests pour $\{\forall x\, \phi(x)\}$ se ramène à celle d'un jeu de tests pour $\{\phi(s),\ s \in S\}$, et le processus se répète récursivement si ϕ contient encore des variables. Sinon,

$T_n = \{\phi(s_0) \wedge \dots \wedge \phi(s_n)\}$ est un jeu de tests acceptable, d'après l'hypothèse de calculabilité sur les symboles de L.

Ces deux types de réduction, adaptés à chaque problème particulier, nous seront, dans les exemples présentés aux chapitres suivants, suffisants pour construire effectivement un jeu de tests acceptable de la forme sus-dite.

5.4.2.3 De l'obtention du contexte

Nous avons donc construit un jeu de tests $\mathcal{T}^1 = \langle H_0 \sqcup H_1, (T_n^1)_{n \in \mathbb{N}^-} \rangle$, sur un contexte virtuel $\mathcal{C}_0' = \langle L_1, S, \emptyset, A \rangle$, avec $L_1 \supseteq L_0$. En appliquant les techniques présentées en 5.1-3., nous pouvons restreindre conservativement $\mathcal{C}_0$, de façon à en faire un contexte sur L_1, puis un $H_0 \sqcup H_1$-contexte, puisque, par construction, $H_0 \sqcup A \vdash H_1$.

Soit $\mathcal{C}_1 = \langle L_1, S, (\mathcal{S}_1), A \rangle$ ce contexte. Nous pouvons faire croître $(\mathcal{S}_1)$ par les techniques d'optimisation présentées en 5.4.2., et éliminer de H_1 les hypothèses inutiles.

Si nous n'avons formulé que des hypothèses "probablement" valides, alors le principe de couplage sera applicable.

H_1 constitue les <u>hypothèses de construction</u>. A la différence de H_0, <u>elles ne sont en général pas validées</u> par la structure représentant l'objet sous test, mais elles le sont cependant "<u>probablement</u>".

5.4.2.4 De l'optimisation

Une fois $\mathcal{C}_1$ obtenu sur $\mathcal{C}_s$, il est possible de chercher à l'améliorer, selon des critères essentiellement <u>subjectifs</u>, par les techniques décrites en 5.3.1. et 5.3.2.

Nous distinguons plusieurs niveaux d'optimisation, selon les hypothèses utilisées par les techniques mises en jeu.

La <u>0-optimisation</u> n'utilise que les hypothèses de H_0, les hypothèses initiales. Celles-ci étant "très probablement vérifiées", elle nous paraît <u>parfaitement légitime</u>

La <u>1-optimisation</u> utilise de plus les hypothèses de construction, H_1, des hypothèses essentiellement techniques "probablement vérifiées". Nous la considérons comme <u>légitime</u>, sous certaines conditions détaillées dans la discussion exposée au paragraphe 5.4.3.

La <u>2-optimisation</u> utilise des <u>hypothèses d'optimisation</u> supplémentaires pour restreindre à nouveau le contexte, et obtenir sur celui-ci un jeu de tests meilleur. Nous considérons une telle optimisation comme <u>suspecte</u>, car elle peut diminuer la validité du principe de couplage. La rentabilité d'une telle optimisation devra être étudiée avec soin dans chaque cas particulier.

5.4.3. Discussion

La présentation précédente n'est bien sûr qu'une ébauche de la méthode qui sera effectivement appliquée aux chapitres suivants. La place de l'intuition y est grande, mais clairement délimitée. Elle induit les hypothèses de construction successives, choisissant la forme (régularité ou uniformité) qui paraît la mieux adaptée au cas étudié. Elle induit les optimisations nécessitées par des critères, subjectifs mais prépondérants en pratiques, qui échappent à la théorie. Plus généralement, elle dispose de l'arsenal de techniques rigoureuses que nous avons développé dans ce chapitre, et qui garantissent que le jeu de tests produit sera conforme aux exigences théoriques.

Remarquons que, de toute façon, quels que soient les choix faits au cours de la phase de construction, le théorème d'équivalence nous assure que tout jeu de test acceptable potentiel comparable à celui effectivement construit est en fait équivalent à celui-ci.

Enfin, nous voulons attirer l'attention du lecteur

sur le fait que de nombreuses hypothèses de construction, notamment celles de régularité, dépendent de paramètres. C'est le cas d'une hypothèse de la forme

$$\{\phi(0) \wedge \ldots \wedge \phi(k) \rightarrow \forall x\, \phi(x)\}$$

Le rôle d'une telle hypothèse est d'assurer que $\{\phi(0), \phi(1), \ldots\}$ est formellement équivalent à $\{\forall x\, \phi(x)\}$, et donc la constante k n'a de valeur que purement technique, étant lié à des problèmes de démonstrabilité et d'induction inhérents à la logique du premier ordre.

Dans la pratique, pour l'utilisateur du test, k est supposé "infini", ou, tout du moins, assez grand pour que cette hypothèse soit une trivialité. Supprimer cette hypothèse revient, dans la plupart des cas, à n'obtenir un jeu de tests que logiquement acceptable, et non formellement acceptable.

Nous nous interdisons donc, à titre de discipline, toute manipulation formelle, et toute optimisation en particulier, faisant intervenir de telles <u>constantes virtuelles</u>.

5.4.4. Conclusion

Cette partie avait pour objet la description critique de la stratégie de construction des jeux de tests présentés aux chapitres suivants.

Les outils et techniques présentés dans ce chapitre nous permettent de considérer la construction d'un jeu de tests sur un contexte comme une activité purement formelle. Les techniques fondamentales en sont

- l'ajout de nouvelles hypothèses (soumises à certaines conditions),
- l'ajout de nouveaux symboles,
- la réduction du problème, par l'invocation du théorème de localisation.

Une fois le jeu de tests construit, nous sommes capable, par l'application de techniques de routine, de reconstruire un contexte adéquat, restriction conservative du contexte initial.

Nous avons de plus proposé certains outils permettant d'améliorer un jeu de tests selon des critères subjectifs, dont la responsabilité est laissée à l'utilisateur. Ces outils sont d'utilisation plus ou moins légitime suivant les hypothèses qu'ils utilisent.

5.5. Conclusion

Ce chapitre nous a permis de présenter, d'un point de vue théorique, les méthodes mises en pratique aux chapitres suivants, sur des exemples concrets.

Nous utilisons principalement deux méthodes pour, à partir d'un contexte initial, construire un contexte admettant un jeu de tests acceptable.

La restriction conservative de contexte permet d'ajouter des symboles au langage, ou de poser des hypothèses supplémentaires. Cependant, si ces hypothèses ne sont pas "probablement vérifiées" par l'objet sous test, le principe de couplage n'est plus applicable.

Le théorème de localisation permet de ramener la construction d'un jeu de tests pour une théorie complexe à celle d'un jeu de tests pour une théorie ne contenant qu'un seul axiome non logique.

L'utilisation cohérente de ces méthodes permet de ramener la phase de construction d'un processus de test à une activité purement formelle.

La stratégie que nous décrivons suppose

que l'on s'intéresse au test d'une propriété A spatiale purement universelle. De plus, nous supposons que le contexte initial $\mathcal{C}_0 = \langle L_0, S, (\mathcal{P}_0), A \rangle$ est tel que L_0 soit un langage dénombrable, et S soit un ensemble dénombrable. Ces conditions, apparemment draconniennes, sont en fait le plus souvent vérifiées, notamment dans le cas du test d'un Σ-axiome sur une Σ-algèbre première, dans le cadre des types abstraits algébriques. De plus, dans ce cas, A ne contient qu'un nombre fini d'axiomes non-logiques (cf. 7.1.4.), et le théorème fondamental du test (cf. 4.4.3.) est applicable.

Chapitre 6 : construction de jeux de tests un exemple.

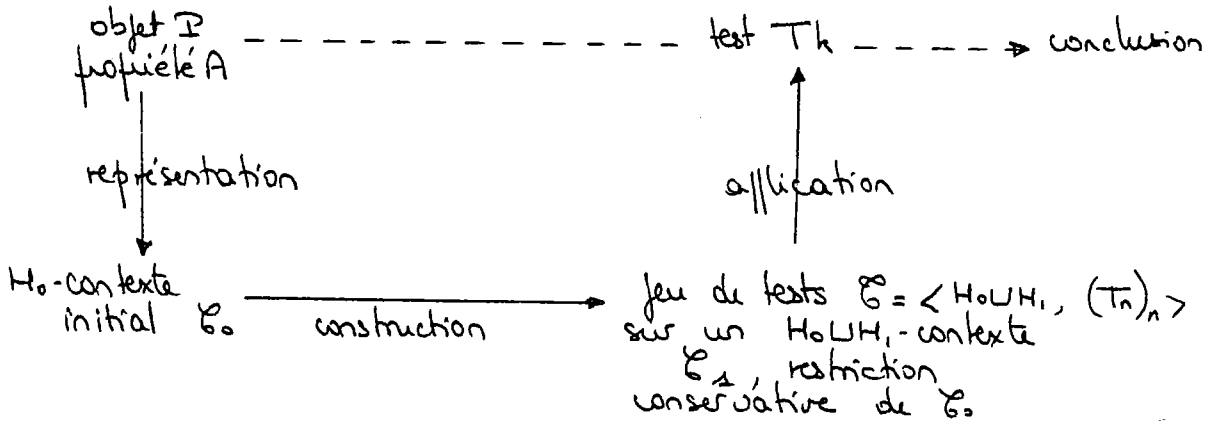
Jusqu'à présent, nous avons exploré, sous un angle théorique, mathématique, la notion de processus de test. Nous en avons donné une définition précise et rigoureuse, dans le cadre de la logique égalitaire du premier ordre. Cette définition nous a permis de dégager les notions de contexte de test et de jeu de tests sur un contexte. Nous avons ensuite étudié les propriétés de ces objets, les ordres dont on peut munir l'ensemble des jeux de tests sur un contexte donné. Nous avons déduit de cette étude, qui pourra sans doute paraître aride, un certain nombre d'outils, et plus généralement de méthodes pratiques de construction de jeux de tests.

L'objet de ce chapitre est de montrer, sur un exemple réaliste, la mise en œuvre concrète de ces résultats. Nous nous intéresserons à un programme de tri. L'on sait la difficulté de concevoir correctement de tels programmes, et le nombre d'erreurs subtiles, erratiques

parfois, que l'on y trouve.

Ce chapitre a été rédigé de telle manière qu'il soit relativement indépendant des précédents, et en particulier du chapitre 5. De nombreuses références seront faites au cours de l'exposé, de manière que le lecteur exigeant puisse confronter notre méthode avec la théorie que nous avons déjà développée.

La notion de processus de test se décompose en trois phases (cf. chapitre 2) :



Dans une première partie, nous présentons, conformément au schéma précédent, le <u>programme sous test</u>, la <u>propriété à tester</u>, le <u>contexte initial</u> $\mathcal{C}_0$ et les <u>hypothèses initiales</u> H_0.

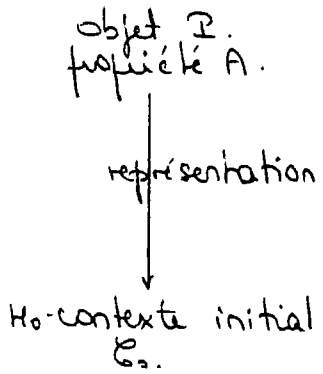
Dans une seconde partie, nous présentons la phase de construction, permettant, à partir de $\mathcal{C}_0$, de produire un <u>contexte</u> $\mathcal{C}_1$, restriction conservative de $\mathcal{C}_0$, et un ensemble H_1 d'<u>hypothèses de</u>

construction, telles que $\mathcal{C}_1$ soit un $H_0 \sqcup H_1$-contexte. Parallèlement, nous exhiberons une suite $(T_n)_{n \in \mathbb{N}}$ de tests telle que $\langle H_0 \sqcup H_1, (T_n)_{n \in \mathbb{N}} \rangle$ soit un jeu de tests acceptable sur $\mathcal{C}_1$.

Dans une troisième partie, nous traiterons de la phase d'application. Nous montrerons les diverses optimisations possibles du jeu de tests obtenu, ainsi que les critères pouvant guider le choix d'un test T_k.

6.1. Phase de représentation

Cette partie montre la réalisation de la première phase du processus de test considéré. Elle se schématise par la flèche suivante.



Ici, P est un programme de tri. A est la propriété correspondant au fait que si les assertions d'entrée sont vérifiées par les données, celles de sortie le sont par les résultats produits par le programme à partir des données. Autrement dit,

$$A = \{\text{entrée}\}\ P\ \{\text{sortie}\}$$

Remarquons que l'on ne s'intéresse qu'à une correction partielle (relative à un système donné d'assertions) de P, et non à une correction intrinsèque, comme le laisse entendre [G&G 75].

Comme le précise la définition proposée en 2.3.4 le H_0-contexte $\mathcal{C}_0$ est la donnée

- d'un langage L_0

- d'un ensemble de base S
- d'une famille $(\mathcal{S}_0)$ de $L_0(S)$-structures
- d'une $L_0(S)$-théorie de propriétés à tester A

et d'une $L_0(S)$-théorie d'hypothèses initiales H_0, telle que toute structure de $(\mathcal{S}_0)$ soit modèle de H_0 (pour les notions de langage, de structure et de théorie, cf. annexe 3).

Intuitivement, $(\mathcal{S}_0)$ est la famille des comportements éventuels du programme. Le comportement réel de celui-ci appartient donc à $(\mathcal{S}_0)$, mais nous ne le connaissons pas plus précisément. H_0 est l'ensemble de certaines hypothèses formelles réglant les comportements éventuels du programme.

6.1.1. Programme et propriété : P, A

Nous empruntons cet exemple à REYNOLDS [Reyn 81]. Le programme suivant trie un tableau de taille n, X[1,n], d'entiers.

```
P:      begin integer m;
          m := n;
          while 1 ≤ m do
            begin integer j;
              begin integer k;
              j := 1 ; k := 1 ;
              while k < m do
                begin k := k+1;
                if X(k) > X(j) then j := k end
              end;
            begin integer t;
              t := X(j); X(j) := X(m); X(m) := t end;
            m := m-1
            end
        end
```

Ce programme a pour but de trier le tableau X en rejetant l'un des maximums de la partie non encore triée au début de la partie déjà triée.

L'assertion d'entrée est

{ X est un tableau d'entiers de longueur n, $n \geq 0$ }

Celle de sortie est

{ X' est un tableau d'entiers de longueur n, $n \geq 0$, permutation de X, trié selon $\leq$ }

Dans cet énoncé, nous avons distingué l'état initial et l'état final du tableau considéré en notant X' au lieu de X le second.

Avec des notations intuitives,

$$A = \forall n \ \forall X \ \text{Entrée}(X, n) \rightarrow \text{Sortie}(X', n)$$

6.1.2. Langage et ensemble de base : L_0, S

La difficulté est ici que nous avons affaire avec un objet typé. P fait intervenir des naturels, des éléments de tableau et des tableaux.

Pour bien distinguer les éléments de tableau des naturels, appelons Z leur ensemble ($Z = \mathbb{Z}$) L'ensemble des tableaux sur Z est alors Z^*.

Nous prendrons donc

$$S = \mathbb{N} \sqcup Z \sqcup Z^* \sqcup \{\perp\}$$

Le langage L_0 doit nous permettre de parler du type des objets. Il contiendra donc des prédicats : entier, élément, tableau, $\perp$.

Il contiendra aussi les symboles apparaissant dans le programme : $\leq$, $<$, $\leq_Z$, $<_Z$, $+$, $-$, 1

Nous indiquons par Z les opérations se rapportant aux éléments pour éviter toute confusion.

De plus, il nous faut pouvoir accéder à un élément d'un tableau, parler de la longueur de celui-ci. Nous prendrons donc des symboles : accès, longueur.

Enfin il nous faudra exprimer que deux tableaux sont permutation l'un de l'autre,

et qu'un tableau est trié. Nous ajoutons les symboles : permutation, trié.

Par ailleurs, nous prenons un symbole F pour exprimer le comportement externe du programme : si x est un tableau, $F(x)$ est le résultat du programme P appliqué à x.

Donc, nous posons

$L_0 = \{$ entier, élément, tableau, $\perp$
$\leq, <, \leq_z, <_z, +, -, \cdot$
accès, longueur
permutation, trié
F $\}$

Remarquons que S et L_0 sont dénombrables (cf. 5.2.2.)

6.1.3. Structures : $(\mathcal{S}_0)$

Grossièrement, nous considérons que, quelleque soit les erreurs contenues dans le programme P, "tout se passe comme si" tous les symboles autres que F étaient implantés correctement.

Plus précisément, $(\mathcal{S}_0)$ est la famille de <u>toutes</u> les $L_0(S)$-structures $\mathcal{S}_0$ telles que les conditions suivantes soient vérifiées.

- $\text{entier}_{\mathcal{S}_0}(x)$ est vrai si et seulement si $x \in \mathbb{N}$. idem pour $\text{élément}_{\mathcal{S}_0}(x)$ avec Z, $\text{tableau}_{\mathcal{S}_0}(x)$ avec Z^*, $\perp_{\mathcal{S}}(x)$ avec $\{\perp\}$
- $\leq, <, +, -, 1$ ont leur signification habituelle sur $\mathbb{N}$, et sont faux si l'un de leurs arguments n'est pas dans $\mathbb{N}$ (ou valent $\perp$ pour $+, -$)
- $\leq_Z, <_Z$ ont leur signification habituelle sur Z ($Z = \mathbb{Z}$), et sont faux en-dehors de leur domaine
- $\text{longueur}_{\mathcal{S}_0}(x)$ est la longueur de x si x est un tableau, $\perp$ sinon
- $\text{accès}_{\mathcal{S}_0}(x, i)$ est le $i^{\text{ème}}$ élément de x si x est un tableau de longueur supérieure ou égale à i (i étant un naturel supérieur ou égal à 1), et vaut $\perp$ dans tous les autres cas
- $\text{permutation}_{\mathcal{S}_0}(x, y)$ est vrai si et seulement x et y sont deux tableaux de même longueur

permutations l'un de l'autre, et faux sinon

- trié (x) est vrai si et seulement si x est un tableau trié selon $\leq_Z$ (et donc en particulier pour les tableaux de longueur 0 ou 1), et faux sinon.

De plus, nous nous restreignons aux structures $\mathcal{S}_0$ telles que $F_{\mathcal{S}_0}$ soit une application calculable sur Z^*, valant $\perp$ ailleurs, telle que pour tout tableau x, $F_{\mathcal{S}_0}(x)$ soit une permutation de x.

Remarquons que ceci suppose que le programme P s'arrête en fournissant une valeur pour toute donnée d'un tableau de longueur ≥ 0, et que cette valeur est celle d'un tableau issu de la donnée par permutation.

Notons enfin que tous les symboles de $\mathcal{L}_0$ ont leur interprétation calculable sur toute structure $\mathcal{S}_0$ de $(\mathcal{S}_0)$ (cf. 3.4.3.2.)

6.1.4 Propriété sous test : A

Intuitivement, nous voulons tester

$$\{\text{Entrée}(x, n)\}\ P\ \{\text{Sortie}(x', n)\}$$

Nous pouvons exprimer cette intuition sous la forme de la formule suivante.

$$\forall n \Big[\text{naturel}(n) \rightarrow$$
$$\forall x \Big[\text{tableau}(x) \wedge \text{longueur}(x) = n \rightarrow$$
$$\text{longueur}(F(x)) = n \wedge$$
$$\text{permutation}(x, F(x)) \wedge$$
$$\text{trié}(F(x)) \Big] \Big]$$

A est la $L_0(s)$ - théorie ayant pour seul axiome non-logique la formule ci-dessus.

A est une formule purement universelle (cf. 3.4.3.2.).

A n'ayant qu'un seul axiome non-logique, nous sommes dans les conditions d'application du théorème fondamental du test (cf. 4.4.3)

6.1.5 Hypothèses initiales : H_0

H_0 contient des formules du premier ordre vérifiées par <u>toutes</u> les structures de $(\mathcal{S}_0)$. Nous pourrions donc prendre comme H_0 l'ensemble de toutes les formules vérifiées conjointement par <u>toutes</u> les structures de $(\mathcal{S}_0)$.

Il nous semble cependant plus intéressant de choisir, parmi ces formules, seulement celles qui nous seront nécessaires lors de la phase de construction.

Pour chaque individu s de $L_0(S)$ appartenant à $\mathbb{N}$, nous prenons la formule

naturel (s)

Par exemple, naturel (0), naturel (1) ...

Par ailleurs, d'après la définition de $(\mathcal{S}_0)$, famille des comportements éventuels de P, nous savons que tous ces comportements vérifient

$$\forall x \left[\text{tableau}(x) \rightarrow \text{permutation}\ (x, F(x)) \right]$$

$$\forall x \left[\text{tableau}(x) \rightarrow \text{longueur}\ (x) = \text{longueur}\ (F(x)) \right]$$

Il est bien évident que, par le choix de $(\mathcal{S}_0)$ (cf. 6.1.3.), toutes ces formules sont vérifiées par toutes les structures de $(\mathcal{S}_0)$.

Nous définissons donc H_0 comme la Lö(S). théorie ayant cette infinité de formules comme seuls axiomes non-logiques. Par la remarque ci-dessus, pour toute structure $\mathcal{S}_0$ de (f_0),

$$\mathcal{S}_0 \models H_0.$$

6.1.6. Conclusion

Nous avons donc défini ainsi, selon la définition proposée en 2.3.4. un H_0-contexte $\mathcal{C}_0 = \langle L_0, S, (\mathcal{P}_0), A \rangle$, à partir du problème concret posé par la donné de l'objet P et de la propriété à tester A.

Comme le lecteur a pu le constater, le passage du problème concret à sa représentation abstraite ne pose aucun problème de difficulté. En particulier, les choix de S, $(\mathcal{P}_0)$ et A découlent directement de la connaissance intuitive, empirique, du problème posé. La connaissance a priori du programme sous test y intervient crucialement.

Les choix de L_0 et H_0 sont plus délicats, du moins apparemment, puisqu'ils répondent à des nécessités mises à jour lors de la phase de construction. Cependant, pour une large part, ils ne sont pas cruciaux. En effet, par les techniques décrites en 5.1.3., il est possible, lors de la phase de construction, d'ajouter de nouveaux symboles et de nouveaux axiomes d'hypothèse, si cela s'avère nécessaire.

6.2. Phase de construction

Cette partie montre la réalisation de la seconde phase du processus de test considéré. Celle-ci est schématisée par la flèche suivante.

$$\begin{array}{c} H_0\text{-contexte initial} \\ \mathcal{C}_0 = \langle L_0, S, (\mathcal{S}_0), A \rangle \end{array} \xrightarrow{\text{construction}} \begin{array}{l} \mathcal{T} = \langle H_0 \sqcup H_1, (T_n)_n \rangle \\ \text{jeu de tests acceptable} \\ \text{sur le } H_0 \sqcup H_1\text{-contexte} \\ \mathcal{C}_1 = \langle L_1, S, (\mathcal{S}_1), A \rangle \\ \text{restriction conservative de } \mathcal{C}_0 \end{array}$$

En général, il n'existe pas de "bon" jeu de tests (jeu de tests acceptable, cf. 3.4.) sur le contexte initial $\mathcal{C}_0$. Ce fait peut avoir deux causes :

- l'insuffisance du langage L_0 pour exprimer les tests nécessaires,
- la trop grande diversité de la famille $(\mathcal{S}_0)$.

Autrement dit, l'on possède trop peu d'informations sur le programme P pour pouvoir construire un "bon" jeu de tests sur ce programme. Il faut donc faire sur P des hypothèses supplémentaires H_1 (<u>hypothèses de construction</u>), afin de restreindre la famille des comportements éventuels de P, $(\mathcal{S}_0)$, en une famille plus homogène, $(\mathcal{S}_1)$.

De même, étendre L_0 en L_1 revient à

se donner des moyens supplémentaires pour décrire et contrôler le comportement de $\mathcal{P}$.

Comme nous l'avons souligné en 2.1.2.3. lors de la description de la notion de processus, et en 5.1., il est fondamental de ne procéder que par <u>restrictions conservatives de contexte</u>. Les nouvelles hypothèses H_1 ne doivent éliminer aucun comportement éventuel $\mathcal{S}_0$ de $\mathcal{P}$ qui vérifie la propriété à tester A. Une condition suffisante pour cela est $H_0 \sqcup A \vdash H_1$ (cf. 5.1.3.). D'autre part, il ne faut poser que des hypothèses "vraisemblables", sinon la notion même de test perd tout son sens (cf. 2.1.4., 5.1.3. et 5.4.2.4.). En effet, même si $\mathcal{P}$ ne vérifie pas réellement H_1, il faudra que "tout se passe comme si" $\mathcal{P}$ les vérifiait

L'outil principal utilisé est <u>le théorème de localisation</u>, présenté en 5.2. Celui-ci permet de ramener, <u>sans perte d'information</u>, le test d'une propriété complexe (éventuellement infinie) à celui d'une propriété simple. Typiquement, si ϕ est une formule, le test de la propriété $\{\phi(n), n \in \mathbb{N}\}$ est ramené à celui de $\{\phi(n)\}$, n étant considéré <u>comme une constante</u>

Partant du contexte $\mathcal{C}_0 = \langle L_0, S, (\mathcal{S}_0), A \rangle$, guidé par les conditions d'applications du théorème de localisation, nous allons étendre peu à peu L_0 et H_0 pour ramener le problème complexe à une famille de problèmes simples, que nous résoudrons de manière générique. Au départ, donc,

$$L_1 := L_0 .$$

$$H_1 := \emptyset$$

$$(\mathcal{S}_1) := (\mathcal{S}_0) .$$

(transitivité de la restriction conservative, cf. 5.1.2.)

6.2.1. Première réduction

Nous cherchons à tester la propriété (cf. 6.1.4.)

$$A = \forall n\ [\text{naturel}(n) \rightarrow \forall x\ [\text{tableau}(x) \wedge \text{longueur}(x) = n \rightarrow \text{longueur}(F(x)) = n \wedge \text{permutation}(x, F(x)) \wedge \text{trié}(F(x))]] \quad (1)$$

Mais H_0 contient les formules (cf. 6.1.5.)

$$\forall x\ [\text{tableau}(x) \rightarrow \text{permutation}(x, F(x))] \quad (2)$$

$$\forall x\ [\text{tableau}(x) \rightarrow \text{longueur}(x) = \text{longueur}(F(x))] \quad (3)$$

Nous voyons donc qu'une partie de A est déjà contenue dans les hypothèses initiales. Il est bien évident qu'il est inutile de tester cette partie, vérifiée a priori par tous les comportements $\mathcal{S}_0$ éventuels de P.

Nous sommes donc ramené à tester la partie restante.

$$B = \forall n\ [\text{naturel}(n) \rightarrow \forall x\ [\text{tableau}(x) \wedge \text{longueur}(x) = n \rightarrow \text{trié}(F(x))]] \quad (4)$$

Il est facile de montrer que tout jeu

de tests acceptable sur le $H_0 \cup H_1$-contexte $\langle L_1, S, (\mathcal{S}_1), B \rangle$ le sera sur le $H_0 \cup H_1$-contexte $\langle L_1, S, (\mathcal{S}_1), A \rangle$, et ce quels que soient L_1 extension de L_0, H_1 et $(\mathcal{S}_1)$.

Le problème du test de A se réduit donc, <u>sans perte d'information</u>, à celui du test de B. (justification complète, cf. 5.2.3.)

6.2.2. Elimination de $\forall n$

Nous sommes donc ramené à chercher un jeu de tests acceptable sur un contexte restreint conservativement à partir du H_0-contexte $< L_1, S, (\mathcal{S}_1), B >$, avec

$$B = \forall n \, [\, \text{naturel}(n) \rightarrow \forall x \, [\, \text{tableau}(x) \wedge \text{longueur}(x) = n \rightarrow \text{trié}(F(x))\,]. \qquad (4)$$

Pour simplifier les écritures, étendons L_0 par l'ajout d'un nouveau symbole b, avec l'axiome :

$$\forall n \, [\, b(n) \leftrightarrow \forall x \, [\, \text{tableau}(x) \wedge \text{longueur}(x) = n \rightarrow \text{trié}(F(x))\,]\,] \qquad (5)$$

Posons donc $L_1 := L_1 \cup \{b\}$

$H_1 := H_1 \sqcup \{(5)\}$

Comme en 6.2.1, sous $H_0 \sqcup H_1$, le test de B est équivalent à celui de

$$\forall n \, [\, \text{naturel}(n) \rightarrow b(n)\,] \qquad (6)$$

que, par abus de notation, nous appelons encore B (pour une justification du procédé plus complète, cf. 5.1.3.).

Intuitivement, naturel (n) n'étant vrai que si $n \in \mathbb{N}$, il est bien évident que le test de B se ramène à celui de

$$\{ b(0), b(1), b(2), \dots \} \qquad (7)$$

Cependant, nous ne pouvons procéder directement car (6) <u>n'est pas formellement équivalent</u> à (7) (cf. annexe 3 et 3.4.3.2.). Il nous faut introduire une nouvelle hypothèse pour assurer l'équivalence de (6) et (7) (conformément à 5.2.3. et 5.4.2.2.).

Cette hypothèse sera une hypothèse de régularité (8)

$$b(0) \wedge b(1) \wedge \dots \wedge b(k) \rightarrow \forall n \, [\text{naturel}(n) \rightarrow b(n)]$$

Examinons (8) de plus près. Si P vérifie A, alors la structure S_0 représentant réellement le comportement de P vérifie (6), et donc (8) trivialement, puisque si β est vrai, $\alpha \rightarrow \beta$ est vrai que α soit vrai ou faux! Si P ne vérifie pas A, soit k_0 la longueur du tableau le plus court mettant en échec les assertions sur P. Alors par définition, S_0 <u>ne valide pas</u> $b(k_0)$, et donc S_0 <u>valide</u> (8) dès que $k \geqslant k_0$, puisque $\alpha \rightarrow \beta$ est vrai si α et β sont faux !

Donc <u>il existe un k_0 assez grand</u> tel que

(8) soit une hypothèse vérifiée par le programme sous test $\mathcal{P}$, mais <u>nous ne connaissons pas</u> ce k_0 (tous les $k \geq k_0$ conviennent aussi).

Mais, paradoxalement peut-être, l'existence d'un tel k_0 suffit à notre tâche. Nous laissons donc dans (8) la constante k indéterminée. Nous appellerons une telle constante une <u>constante virtuelle</u>.

Sous cette hypothèse (8), quelle que soit la valeur de k, (7) est équivalent à (6). Le test de B se ramène donc à celui de

$$\{ b(0), b(1), b(2), \dots \} \qquad (7)$$

avec $\qquad H_1 := H_1 \sqcup \{(8)\}$.

Par le théorème de localisation (cf. 5.2), nous sommes ramené à celui de

$$\{ b(n) \}, \qquad (9)$$

où n doit être considérée comme une constante vérifiant naturel (n).

Nous avons donc éliminé le quantificateur universel $\forall n$.

6.2.3. Elimination de $\forall x$: première stratification

Nous sommes donc ramené au problème de la recherche d'un jeu de tests acceptable sur le $H_0 \sqcup H_1$ - contexte

$$\langle L_1, S_1(\mathcal{S}_1), C \rangle , \qquad (10)$$

avec

$$C = \{ b(n) \} , \qquad (11)$$

n étant une constante vérifiant l'axiome

$$\text{naturel}(n) \qquad (12).$$

Par l'axiome (5) de H_1, puisque nous travaillons sur un $H_0 \sqcup H_1$ - contexte, nous sommes ramené au test de

$$\forall x \, [\, \text{tableau}(x) \wedge \text{longueur}(x) = n \longrightarrow \text{trie}(F(x)) \,] \qquad (13)$$

que nous notons, par abus de notation, aussi C.

Nous voulons éliminer ce $\forall x$. Pour cela, l'ensemble des tableaux de longueur n n'étant pas naturellement ordonné, nous ne pouvons appliquer la méthode du 6.2.2. Aussi, selon 5.4.2.2., et 2.4.2.2., nous subdivisons l'ensemble des x vérifiant tableau(x) $\wedge$ longueur(x) = n en sous-domaines supposés intuitivement "assez uniformes" par rapport

à la propriété à tester. Un domaine est dit uniforme par rapport à une propriété, si tous les éléments du domaine vérifient la propriété, ou aucun ne la vérifie (cf. annexe 1).

Enrichissons donc le langage de nouveaux symboles :

$$U_\sigma \quad , \sigma \in \mathcal{S}_n \tag{14}$$

($\mathcal{S}_n$ est le groupe des permutation de n éléments, d'ordre $n!$).

$$L_1 := L_2 \sqcup \{ U_\sigma , \sigma \in \mathcal{S}_n \} \tag{15}$$

Etendons chacune des structures de $(\mathcal{S}_1)$ en donnant à U_σ la signification suivante. $U_{\sigma_{\mathcal{S}_1}}(x)$ est vrai, si et seulement si x est un tableau de longueur n, noté $(x_1 \dots x_n)$ tel que $x_{\sigma(1)} \leq_{z_{\mathcal{S}_1}} x_{\sigma(2)} \leq_{z_{\mathcal{S}_1}} \cdot \leq_{z_{\mathcal{S}_1}} x_{\sigma(n)}$.

Il est bien évident que si x est un tableau de longueur n, il existe un (éventuellement plusieurs) σ tel que $U_{\sigma_{\mathcal{S}_1}}(x)$.

Donc, toute structure $\mathcal{S}_1$ de $(\mathcal{S}_1)$ valide la formule :

$$\forall x \left[\text{tableau}(x) \wedge \text{longueur}(x) = n \leftrightarrow \bigvee_{\sigma \in \mathcal{S}_n} U_\sigma(x) \right] \tag{16}$$

Nous pouvons donc rajouter cet axiome à H_1 sans modifier $(\mathcal{S}_1)$

$$H_1 := H_1 \sqcup \{(16)\}$$

(16) exprime que les sous-domaines U_σ recouvrent le domaine des tableaux de longueur n, mais ne forment pas, en général, une partition de ce domaine ; par exemple, tout tableau $(x_1 \cdots x_n)$, avec $x_1 = \cdots = x_n$ appartient à tous les U_σ.

Sous l'axiome (16), maintenant dans H_1, (13) est équivalent à :

$$\bigwedge_{\sigma \in S_n} \forall x \left[\text{tableau}(x) \wedge \text{longueur}(x) = n \wedge U_\sigma(x) \rightarrow \text{Trié}(F(x)) \right]. \qquad (17).$$

Nous sommes donc ramené au test de l'ensemble de formules suivant (qui est fini !) :

$$\left\{ \forall x \left[\text{tableau}(x) \wedge \text{longueur}(x) = n \wedge U_\sigma(x) \rightarrow \text{Trié}(F(x)) \right], \ \sigma \in S_n \right\}. \qquad (18)$$

Par le théorème de localisation, nous sommes ramené au test d'une seule formule :

$$\forall x \left[\text{tableau}(x) \wedge \text{longueur}(x) = n \wedge U_\sigma(x) \rightarrow \text{Trié}(F(x)) \right], \qquad (19)$$

σ étant supposé fixé.

Nous n'avons pas éliminé $\forall x$, mais nous avons restreint virtuellement son extension aux x de U_σ (cf. [Demo 82], [Nico 79] pour des problématiques voisines en bases de données)

6.2.4. Elimination de $\forall x$: seconde stratification

Nous sommes donc ramené à la recherche d'un jeu de tests acceptable sur le contexte :

$$< L_1, S, (\mathcal{S}_i), D > \qquad (20)$$

avec

$$D = \{ \forall x \, [\text{tableau}(x) \wedge \text{longueur}(x) = n \wedge U_\sigma(x) \rightarrow \text{trié}(F(x))] \}, \qquad (19)$$

n étant une constante telle que naturel(n), et σ un indice appartenant à $\mathcal{S}_n$.

Intuitivement, D signifie que P vérifie ses assertions sur le sous-domaine des tableaux de longueur n, ordonnés selon U_σ.

Conformément à 5.4.2.2., examinons l'uniformité de ce sous-domaine. Sachant que P vérifie ses assertions pour un tableau x_0 de longueur n, et tel que $U_\sigma(x_0)$, peut-on <u>raisonnablement induire</u> que P les vérifie pour tout tableau x de ce type ?

Autrement dit, si P "marche" pour le tableau $(-4 \;\; 2 \;\; 1)$, peut-on induire raisonnablement qu'il marche pour $(-3872 \; -683 \; -2222)$?

Si notre sentiment est qu'une telle induction est raisonnable, alors nous rajoutons l'hypothèse d'uniformité suivante.

$$[\exists x\ (\text{tableau}(x) \wedge \text{longueur}(x)=n \wedge U_\sigma(x) \wedge \text{trié}(F(x)))] \rightarrow$$
$$[\forall x\ (\text{tableau}(x) \wedge \text{longueur}(x)=n \wedge U_\sigma(x)) \rightarrow \text{trié}(F(x))] \qquad (21)$$

(21) se prononce de la manière suivante : "Si le programme P vérifie ses assertions <u>pour un x</u> tel que tableau(x), longueur(x)=n et $U_\sigma(x)$, alors il les vérifie <u>pour tous les x</u> de ce type."

Mais, en 6.2.3., nous avons remarqué que le tableau de longueur n (0 ... 0) appartient à tous les U_σ. Le fait que P "marche" pour (0 ... 0) est-il suffisant pour induire raisonnablement qu'il "marche" pour tous les tableaux de longueur n ? Oui si n = 0 ou 1, mais pour $n \geq 2$?

Certes non, et c'est pourquoi il nous faut opérer une seconde stratification du domaine. La première faisait intervenir le symbole $\leq_E$, alors que le programme opère avec le symbole $<_E$. Il est donc naturel d'essayer d'utiliser à présent ce symbole.

Soit $x = (x_1 \dots x_n)$. $U_\sigma(x)$ signifie, d'après 6.2.3, $x_{\sigma(1)} \leq_E \dots \leq_E x_{\sigma(n)}$. Divisons ce domaine en mettant en évidence les égalités et inégali-

tés. Nous sommes donc amené à introduire de nouveaux symboles $V\psi$. Intuitivement, ψ est une suite de $n-1$ symboles, chacun d'eux étant soit "=", soit "$\neg$=" ("$\neg$" est le symbole de négation). $V\psi(x)$ sera donc, par exemple, de la forme suivante, pour $\psi = ($"=", "$\neg$=", ..., "=", "$\neg$=")

$$(x_{\sigma(1)} = x_{\sigma(2)}) \wedge \neg(x_{\sigma(2)} = x_{\sigma(3)}) \wedge \dots \wedge (x_{\sigma(n-2)} = x_{\sigma(n-1)}) \wedge \neg(x_{\sigma(n-1)} = x_{\sigma(n)}) \qquad (22)$$

Reprenons donc tout cela plus formellement.

Etendons L_1 à l'aide des symboles $V\psi$, pour ψ compris entre 0 et $2^{n-1}-1$ (puisque il existe 2^{n-1} suites distinctes de $n-1$ symboles choisis dans $\{$"=", "$\neg$="$\}$).

$$L_2 := L_1 \cup \{ V\psi, 0 \leq \psi \leq 2^{n-1}-1 \} \qquad (23)$$

Ceci, bien sûr, n'est valable que si $n \geq 2$. Dans le cas $n=0$ et $n=1$, le problème ne se pose pas.

Conformément à 5.1.3., étendons chacune des structures de $(\mathcal{S}_1)$ en donnant à chaque $V\psi$, $0 \leq \psi \leq 2^{n-1}-1$ la signification déduite de la formule (22) sur le domaine des tableaux de longueur n vérifiant $U\sigma$. $V\psi$ est défini de plus comme étant faux

en dehors de ce domaine.

Si x est un tableau de longueur n tel que $U_\sigma(x)$, il existe un ψ tel que $V_\psi(x)$. Autrement dit, les V_ψ opèrent un recouvrement (une partition en fait) de ce domaine.

Toutes les structures $\mathcal{S}_1$ de $(\mathcal{S}_1)$ vérifient donc la formule :

$$\forall x \left[\text{tableau}(x) \wedge \text{longueur}(x) = n \wedge U_\sigma(x) \leftrightarrow \bigvee_{0 \leq \psi \leq 2^{n-1}-1} V_\psi(x) \right]. \tag{24}$$

Nous pouvons donc rajouter cet axiome à H_1 sans modifier $(\mathcal{S}_1)$:

$$H_1 := H_1 \sqcup \{(24)\}.$$

Par (24), (19) est maintenant équivalent à :

$$\bigwedge_{0 \leq \psi \leq 2^{n-1}-1} \forall x \left[\text{tableau}(x) \wedge \text{longueur}(x) = n \wedge U_\sigma(x) \wedge V_\psi(x) \rightarrow \text{trié}(F(x)) \right] \tag{25'}$$

Nous sommes donc ramené au test de l'ensemble (fini) de formules suivant :

$$\left\{ \forall x \left[\text{tableau}(x) \wedge \text{longueur}(x) = n \wedge U_\sigma(x) \wedge V_\psi(x) \rightarrow \text{trié}(F(x)) \right], 0 \leq \psi \leq 2^{n-1}-1 \right\} \tag{25}$$

Par le théorème de localisation, nous sommes ramené au test d'une seule formule :

$$\forall x \, [\, \text{tableau}(x) \wedge \text{longueur}(x) = n \wedge U_\sigma(x) \wedge V_\psi(x) \rightarrow \text{trié}\,(F(x))\,], \qquad (27)$$

l'indice ψ étant supposé fixé, $0 \leq \psi \leq 2^{n-1} - 1$.

Ainsi, par rapport à (19), nous avons restreint encore plus le champ de variation du quantificateur universel $\forall x$.

6.2.5. Elimination de $\forall x$: hypothèse d'uniformité

Nous sommes donc ramené à la recherche d'un jeu de tests acceptable sur le contexte :

$$< L_1 , S, (\mathcal{P}_1), E > \qquad (28)$$

avec

$$E = \{ \forall x [\text{tableau}(x) \wedge \text{longueur}(x) = n \wedge U_\sigma(x) \wedge V_\psi(x) \rightarrow \text{trié}(F(x))] \} \qquad (29)$$

où n est une constante vérifiant naturel(n) $(n \in \mathbb{N}^*)$, σ un indice appartenant à $\mathcal{P}_n$, ψ un indice compris entre 0 et $2^{n-1}-1$ (si $n \geq 2$, sinon l'introduction de V_ψ n'est pas nécessaire, cf. 6.2.3.).

Notre sentiment, contrairement à celui exprimé en 6.2.4., est que le domaine défini par :

$$\text{tableau}(x) \wedge \text{longueur}(x) = n \wedge U_\sigma(x) \wedge V_\psi(x)$$

peut être considéré comme uniforme vis à vis de la formule trié$(F(x))$.

Ce sentiment n'est bien sûr qu'intuitif, empirique, subjectif. Cependant, nous pouvons esquisser quelques justifications.

La plus convaincante est sans doute la remarque suivante. Tous les éléments de ce domaine suivent le même chemin à travers le programme sous test P. Par ailleurs, de manière sans doute moins convaincante, la propriété "trié" n'est liée qu'à la propriété "$\leq_Z$" entre les éléments du tableau considéré ; notre domaine est tel que les éléments des tableaux lui appartenant vérifient entre eux les mêmes inégalités strictes ou égalités ; il est donc "très" uniforme vis à vis de "$\leq_Z$".

Il ne nous est pas possible de justifier plus précisément ce sentiment en l'absence d'une notion absolue de qualité d'un test (cf. la conclusion de cette thèse). Tout ce que nous pouvons affirmer est que <u>l'hypothèse suivante est "probablement" vérifiée par la structure représentant réellement le comportement du programme</u>.

(30)

$$[\exists x\ (\text{tableau}(x) \wedge \text{longueur}(x) = n \wedge U_\sigma(x) \wedge V_\psi(x) \wedge \text{trié}(F(x)))] \rightarrow$$
$$[\forall x\ (\text{tableau}(x) \wedge \text{longueur}(x) = n \wedge U_\sigma(x) \wedge V_\psi(x) \rightarrow \text{trié}(F(x)))]$$

(30) exprime que si P "marche" pour <u>un tableau du domaine</u>, P "marche" pour <u>tous les tableaux de ce domaine</u>.

Remarquons que si une structure $\mathcal{S}_0$ représentant un comportement éventuel de P vérifie la propriété sous test A (cf. 6.1.4.)

$$\begin{array}{l} \forall n \, [\, \text{natural}(n) \rightarrow \\ \quad \forall x \, [\, \text{tableau}(x) \wedge \text{longueur}(x) = n \rightarrow \\ \qquad \text{longueur}(F(x)) = n \wedge \\ \qquad \text{permutation}(x, F(x)) \wedge \\ \qquad \text{trié}(F(x)) \,]\,] \end{array} \qquad (1)$$

alors elle vérifie (30).
Plus précisément $A \vdash \{(30)\}$.
Par 5.1.3., nous avons le droit d'ajouter (30) à H_1, et de réduire $(\mathcal{S}_1)$ en conséquence.

$$H_1 := H_1 \sqcup \{(30)\}$$
$$(\mathcal{S}_1) := \{ \mathcal{S}_1 \in (\mathcal{S}_1),\ \mathcal{S}_1 \models (30) \}$$

Nous ne conservons donc que les structures vérifiant la propriété (30).
Il est possible que, ce faisant, nous ne conservions pas la structure $\mathcal{S}_1$ représentant <u>réellement</u> le comportement de P. Mais celle-ci vérifiant "probablement" (30), par <u>le principe du couplage</u> (cf. 2.1.4.), "tout se passera comme si"

elle avait été conservée, et les conclusions du test seront "probablement" correctes.

L'estimation (subjective) de ces "probables" est sans doute le point le plus délicat de la méthode. La qualité de la conclusion du test est en fait directement fonction de ce "probable".

6.2.6. Production d'un jeu de tests

Pour produire un jeu de tests acceptable, il nous suffit maintenant d'exécuter le programme P sur une valeur quelconque du domaine considéré. Si les assertions sont vérifiées, (30) nous permet alors de conclure que P "marche" sur le domaine tout entier.

Plus précisément, ajoutons à L_1 un symbole de constante t.

$$L_1 := L_1 \cup \{ t \} \qquad (31)$$

Etendons les structures de $(\mathcal{S}_1)$ de toutes les manières possibles telles que t soit bien un individu tel que

$$\begin{aligned} &\text{tableau}(t) \\ &\text{longueur}(t) = n \\ &U_\sigma(t) \\ &V_\psi(t). \end{aligned} \qquad (32)$$

Nous obtenons une famille de structures que nous notons encore $(\mathcal{S}_1)$ (cf. 5.1.3. pour une justification complète).

Par ces choix, toutes les structures de $(\mathcal{S}_1)$ vérifient la formule :

$$\text{tableau}(t) \wedge \text{longueur}(t) = n \wedge U_\sigma(t) \wedge V_\psi(t) \qquad (33)$$

Sans modifier $(\mathcal{S}_1)$, nous pouvons donc ajouter à H_1.

$$H_1 := H_1 \sqcup \{(33)\} \qquad (34)$$

Alors $\langle H_0 \sqcup H_1, (T_i)_{n \in \mathbb{N}} \rangle$,

avec

$$T_i = \{ \text{trié}(F(t)) \} \qquad i \in \mathbb{N} \qquad (35)$$

est un <u>jeu de tests acceptable sur le contexte $\langle L_1, S, (\mathcal{S}_1), E \rangle$</u>. (28)

En effet,

$\{\text{trié}(F(t))\} \sqcup \{(33)\}$

démontre (36)

$\exists x \, [\, \text{tableau}(x) \wedge \text{longueur}(x) = n \wedge U_\sigma(x) \wedge V\psi(x)$
$\wedge \, \text{trié}(F(x))\,]$

et $\{(36)\} \sqcup \{(30)\}$ démontre E,

ce qui prouve que $\mathcal{C}$ est un jeu de tests <u>asymptotiquement formellement valide</u> (cf. 3.2.2)

D'autre part,

$E \sqcup \{(33)\}$ démontre $\{\text{trié}(F(t))\}$,

ce qui montre que $\mathcal{C}$ est un jeu de tests <u>formellement non biaisé</u> (cf. 3.3.1)

Enfin, T_i est bien un test, puisque les symboles "trié" et "t" sont d'interprétations calculables sur toutes les structures de $(\mathcal{S}_1)$. (cf. 2.3.5. et 6.1.3).

6.2.7. Description du jeu de tests et du contexte produits : $\mathcal{C}_1$ et $\mathcal{C}_1$

Résumons maintenant le résultat de cette phase de construction en rassemblant les divers sous-problèmes définis par le théorème de localisation.

Ceci nous conduit à faire varier ensemble n, σ et ψ. Pour distinguer les divers symboles introduits à n fixé, nous leur ajouterons l'indice n. De même, ceux introduits à n et σ fixés seront surchargés du couple d'indices n, σ, et ceux introduits à n, σ, ψ fixés du triplet n, σ, ψ. Ainsi nous considérerons les symboles $U_{n,\sigma}$ (cf. 6.2.3), $V_{n,\sigma,\psi}$ (cf. 6.2.4), $t_{n,\sigma,\psi}$ (cf. 6.2.6).

Posons :

$$\mathcal{C}_1 = \langle L_1, S, (\mathcal{S}_1), A \rangle$$

$$\mathcal{C}_1 = \langle H_0 \cup H_1, (T_i)_{i \in \mathbb{N}} \rangle$$

S est décrit en 6.1.2.. C'est un ensemble dénombrable.

A est décrit en 6.1.4. C'est une formule (une théorie, en fait) purement universelle.

H_0 est décrit en 6.1.5. C'est la théorie des <u>hypothèses initiales</u>

Examinons maintenant L_1, H_1, $(\mathcal{S}_1)$, $(T_i)_{i \in \mathbb{N}}$.

6.2.7.1 Langage : L_1

L_1 est obtenu en ajoutant à L_0 (cf. 6.1.2) les symboles :

b		(cf. 6.2.2, (5))
$U_{n,\sigma}$	$n \in \mathbb{N}$, $\sigma \in \mathcal{S}_n$	(cf. 6.2.3., (15))
$V_{n,\sigma,\psi}$	$n \geq 2$, $\sigma \in \mathcal{S}_n$, $0 \leq \psi \leq 2^{n-1}-1$	(cf. 6.2.4., (23))
$t_{n,\sigma,\psi}$	$n \geq 2$, $\sigma \in \mathcal{S}_n$, $0 \leq \psi \leq 2^{n-1}-1$	(cf. 6.2.6, (31))

Rappelons que $\mathcal{S}_n$ est le groupe des permutations de n éléments, de cardinal $n!$.

L_1 est donc un langage dénombrable (puisque L_0 l'était).

6.2.7.2 hypothèses de construction : H_1

H_1 est la $L_1(S)$-théorie ayant pour seuls axiomes non-logiques les $L_1(S)$-formules suivantes.

$$\forall n \, [\, b(n) \leftrightarrow \forall x \, [\, \text{tableau}(x) \wedge \text{longueur}(x) = n \rightarrow \text{trié}(F(x))\,]\,] \qquad \text{(cf. 6.2.2, (5))}$$

$b(0) \wedge b(1) \wedge \ldots \wedge b(k) \longrightarrow \forall n\,[\text{naturel}(n) \rightarrow b(n)]$

où k est <u>un certain entier</u> positif, probablement assez grand (cf. 6.2.2, (8))

$$\forall x\,[\,\text{tableau}(x) \wedge \text{longueur}(x) = n \longleftrightarrow \bigvee_{\sigma \in \mathcal{S}_n} U_\sigma(x)\,]$$

pour chaque $n \in \mathbb{N}^-$ (cf. 6.2.3, (16))

$$\forall x\,[\,\text{tableau}(x) \wedge \text{longueur}(x) = n \wedge U_{n,\sigma}(x) \longleftrightarrow \bigvee_{0 \leq \psi \leq 2^{n-1}-1} V_{n,\sigma,\psi}(x)\,]$$

pour chaque $n \geq 2$, $\sigma \in \mathcal{S}_n$ (cf. 6.2.4., (24))

$$[\exists x\,(\text{tableau}(x) \wedge \text{longueur}(x) = n \wedge U_{n,\sigma}(x) \wedge V_{n,\sigma,\psi}(x) \wedge \text{trié}(F(x)))\,] \rightarrow$$
$$[\forall x\,(\text{tableau}(x) \wedge \text{longueur}(x) = n \wedge U_{n,\sigma}(x) \wedge V_{n,\sigma,\psi}(x) \rightarrow \text{trié}(F(x)))\,]$$

pour chaque $n \geq 2$, $\sigma \in \mathcal{S}_n$, $0 \leq \psi \leq 2^{n-1}-1$

(cf. 6.2.5, (30))

$$[\exists x\,(\text{tableau}(x) \wedge \text{longueur}(x) = n \wedge U_\sigma(x) \wedge \text{trié}(F(x))]$$
$$\rightarrow [\forall x\,(\text{tableau}(x) \wedge \text{longueur}(x) = n \wedge U_\sigma(x) \rightarrow \text{trié}(F(x))]$$

pour $n = 0$ ou 1, $\sigma \in \mathcal{S}_n$ (mais $|\mathcal{S}_n| = 1$ alors !)

(cf. 6.2.4)

$$\text{tableau}(t_{n,\sigma,\psi}) \wedge \text{longueur}(t_{n,\sigma,\psi}) = n \wedge U_{n,\sigma}(t_{n,\sigma,\psi}) \wedge V_{n,\sigma,\psi}(t_{n,\sigma,\psi})$$

pour $n \geq 2$, $\sigma \in \mathcal{S}_n$, $0 \leq \psi \leq 2^{n-1}-1$ (cf. 6.2.6, (33))

tableau $(t_{n,\sigma}) \wedge$ longueur $(t_{n,\sigma}) = n \wedge$
$U_{n,\sigma}(t_{n,\sigma})$
pour $n = 0$ ou 1, $\sigma \in \mathcal{S}_n$ (cf. 6.2.4.)

6.2.7.3 Structures : $(\mathcal{S}_1)$

$(\mathcal{S}_1)$ est la famille des $L_1(S)$-structures telles que les symboles de $L_0 \setminus \{F\}$ aient leur signification "habituelle" (cf. 6.1.3.), les symboles b, $U_{n,\sigma}$, $V_{n,\sigma,\psi}$ ayant la signification proposée en 6.2.2, 6.2.3, 6.2.4., et les symboles $t_{n,\sigma,\psi}$ et $t_{n,\sigma}$ vérifiant les axiomes décrits en 6.2.7.2 :

tableau $(t_{n,\sigma,\psi}) \wedge$ longueur $(t_{n,\sigma,\psi}) = n \wedge U_{n,\sigma}(t_{n,\sigma,\psi})$
$\wedge V_{n,\sigma,\psi}(t_{n,\sigma,\psi})$
pour $n \geq 2$, $\sigma \in \mathcal{S}_n$, $0 \leq \psi \leq 2^{n-1}-1$
et tableau $(t_{n,\sigma}) \wedge$ longueur $(t_{n,\sigma}) \wedge U_{n,\sigma}(t_{n,\sigma})$.

De plus, les structures $\mathcal{S}_1$ de $(\mathcal{S}_1)$ doivent être telles que F soit d'interprétation calculable sur Z^*, valant $\perp$ ailleurs, telle que pour tout tableau x, $F_{\mathcal{S}_1}(x)$ soit un tableau issu de x par permutation

Enfin, les structures $\mathcal{S}_1$ de $(\mathcal{S}_1)$ doivent valider la théorie des hypothèses de construction H_1, et notamment les hypothèses de régularité

et d'uniformité (8) et (30).

Remarquons que, en toute rigueur, selon 5.3.2., nous aurions pu considérer la famille de toutes les $L_1(S)$-structures validant H_0 et H_1, et telles que l'interprétation des symboles de L_1 soit calculable. Mais cette famille est trop "grande" par rapport à $(\mathcal{F}_0)$, et bien qu'elle conduise à un contexte tel que le jeu de tests soit acceptable, ce contexte <u>n'est pas une restriction</u> du contexte initial $\mathcal{C}_0$.

6.2.7.4. Tests : $(T_i)_{i \in \mathbb{N}}$

Il est facile de vérifier que l'application du théorème de localisation (cas infini, puis cas fini) (cf. 5.2.2. et 5.2.3) conduit au résultat suivant :

$$T_i = \bigsqcup_{\substack{n, \sigma, \psi \\ n \leq i \\ \sigma \in S_n \\ 0 \leq \psi \leq 2^{n-1}-1 \text{ si } n \geq 2}} T_{n, \sigma, \psi, i} \qquad (\text{cf. } 6.2.6.)$$

De manière plus explicite,

$T_0 = \{ \text{trié}(F(t_0)) \}$ longueur$(t_0) = 0$

$T_1 = T_0 \sqcup \{ \text{trié}(F(t_1)) \}$ longueur$(t_1) = 1$

$$T_2 = T_1 \sqcup \{ \text{trié}(F(t_{2,\sigma_0,0})) \\ \text{trié}(F(t_{2,\sigma_0,1})) \\ \text{trié}(F(t_{2,\sigma_1,0})) \\ \text{trié}(F(t_{2,\sigma_1,1})) \}$$

etc.

T_i est donc formé d'une suite d'expériences de la forme trié$(F(t))$ où t varie sur un certain nombre de constantes.

Intuitivement ces constantes représentent des tirages "aléatoires" de tableaux ayant certaines caractéristiques.

Supposons un tel tirage "aléatoire" réalisé (ce qui revient à fixer une structure S_1) et décrivons la suite de tableaux envisagés par T_3

$()$	t_0, σ_0	$n=0$
(23)	t_1, σ_1	$n=1$
$(5\ \ 5)$	$t_2, \sigma_0, 0$	$n=2$
$(-4\ \ 12)$	$t_2, \sigma_0, 1$	
$(-20\ \ -20)$	$t_2, \sigma_1, 0$	
$(45\ \ 13)$	$t_2, \sigma_1, 1$	
$(0\ \ 0\ \ 0)$	$t_3, \sigma_0, 0$	$n=3$
$(2\ \ 2\ \ 92)$	$t_3, \sigma_0, 1$	
$(-8\ \ -7\ \ -7)$	$t_3, \sigma_0, 2$	
$(-3\ \ 6\ \ 14)$	$t_3, \sigma_0, 3$	
$(9\ \ 9\ \ 9)$	$t_3, \sigma_1, 0$	
$\vdots$	$\vdots$	
$(33\ \ 15\ \ -2)$	$t_3, \sigma_5, 3$	

Plus généralement, T_i contient

$$1 + 1 + \sum_{n=2}^{i} n!\, 2^{n-1} \quad \text{pour } i \geq 2$$

Nous insistons tout particulièrement sur le fait que la notion de tableau "aléatoire" est totalement subjective et extérieure à la théorie. L'hypothèse d'uniformité (30) (cf. 6.2.5.) qui est la clé de voûte de notre travail présent n'impose aucune condition de "représentativité" pour le x du "$\exists x$".
Autrement dit,

$$(1 \quad 2 \quad 3 \quad 4 \quad 5)$$

est un tableau <u>tout aussi "aléatoire"</u> que

$$(-66 \quad -54 \quad 13 \quad 25 \quad 99)$$

pour tester P sur le sous-domaine $V_{5, \sigma_0, 15}$, et le jeu de tests obtenu sera <u>tout aussi acceptable</u> !

Ce point est sans doute le plus paradoxal, mais aussi le plus puissant de notre thèse. Nous aurons l'occasion d'y revenir.

Nous invitons le lecteur à le méditer et à s'en convaincre. L'acceptabilité d'un jeu de tests n'est pas liée au caractère plus ou moins "aléatoire" des expériences qui le composent, mais au <u>système d'axiomes</u> et de <u>règles de déduction formelles</u> qui le sous-

tendent.

Autrement dit, il n'y a aucune raison que 0 soit intrinsèquement un test moins bon que 865 ou que -53. Toute la substance de notre thèse est là.

6.2.8 Conclusion

Cette seconde partie nous a permis de décrire, dans le cas précis de notre exemple, la phase de construction du processus de test.

H_0 - contexte initial $\mathcal{C}_0$ $\langle L_0, S, (\mathcal{S}_0), A \rangle$ $\xrightarrow{\text{construction}}$ jeu de tests $\mathcal{T}$ acceptable sur $\mathcal{C}_1$, $H_0 \cup H_1$ - contexte $\langle L_1, S, (\mathcal{S}_1), A \rangle$, restriction conservative de $\mathcal{C}_0$

Notre démarche a suivi étroitement la méthode, décrite en 5.4.2., consistant à réduire de plus en plus le problème de la recherche d'un jeu de tests acceptable sur un contexte par des utilisations répétées du théorème de localisation. Celles-ci nécessitent l'ajout d'hypothèses de construction et de nouveaux symboles, éventuellement. Procédant sous les conditions décrites en 5.1.3., nous sommes certain de n'effectuer que des restrictions conservatives de contexte, et par transitivité (cf. 5.1.2.) d'obtenir finalement un contexte $\mathcal{C}_1$ restriction conservative de $\mathcal{C}_0$.

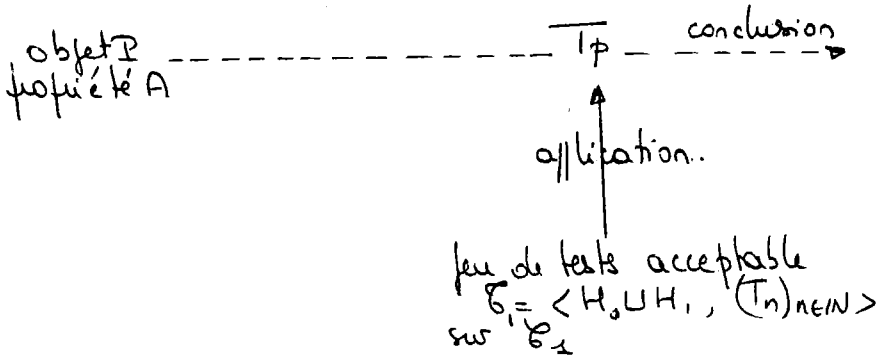
Une large part a été laissée à l'intuition dans la conduite de cette phase du processus de test. Mais celle-ci est, à tout instant,

orientée et contrôlée par les divers théorèmes et propriétés démontrées aux chapitres précédents.

L'application d'un méthode rigoureusement contrôlée par une solide théorie sous-jacente nous a permis de construire un jeu de tests de manière entièrement abstraite et formelle, par la simple manipulation de langages et de théories, en dehors de toute référence à la notion de "valeur aléatoire représentative" qui masque habituellement les processus d'induction (hypothèse de régularité et d'uniformité) rigoureux sous-jacents à la notion de test. Notre approche donne à ces processus un rôle essentiel, rejetant le choix de valeurs réelles de test à la phase d'application.

6.3. Phase d'application

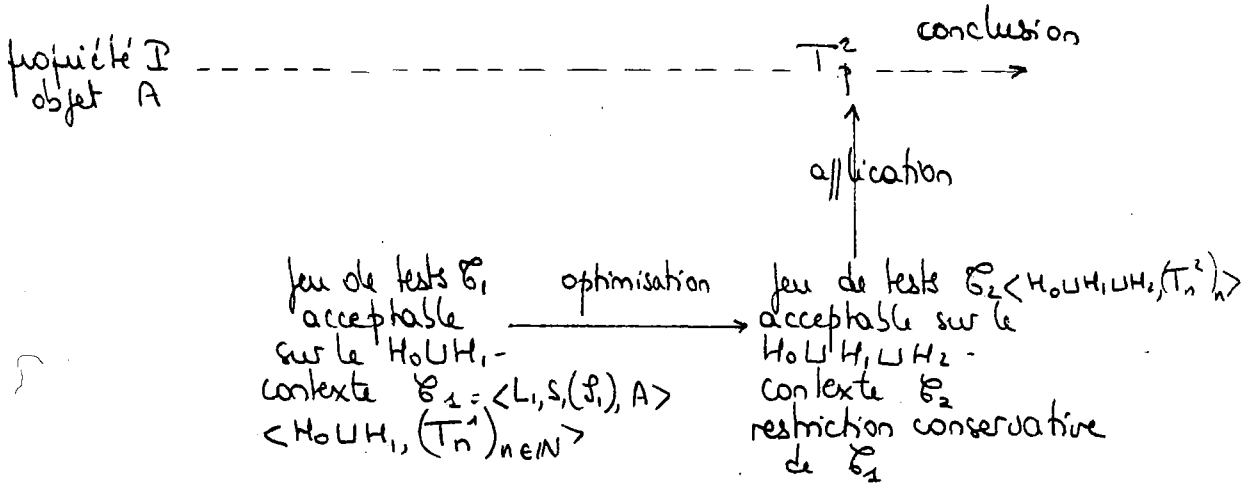
La troisième phase d'un processus de test est la phase d'application (cf. 2.1.3). Elle peut se décrire par le schéma suivant



Elle consiste essentiellement à choisir un indice p tel que le test T_p du jeu de tests $\mathcal{T}$ soit de <u>rentabilité satisfaisante</u> et d'un <u>coût abordable</u> (cf. 2.1.3.). Ce problème est donc extérieur à notre propos. Nous avons proposé en 6.2.7.4. une estimation du nombre d'expériences élémentaires (exécution du programme sur une donnée) composant T_n, et une description de T_3, des valeurs "aléatoires" ayant été données aux constantes.

En général, cependant, il n'existe pas d'indice p satisfaisant (bien que le jeu de tests soit formellement acceptable) par rapport aux exigences de l'utilisateur. Celui-ci cherche alors à optimiser, à améliorer selon

ses propres critères, le jeu de tests proposé.
Ceci se traduit par le schéma suivant.



Nous avons décrit en 5.3. plusieurs outils pouvant réaliser de telles améliorations. Ces outils peuvent être utilisés de différentes manières. L'optimisation peut se fonder sur les hypothèses initiales H_0, auquel cas elle est parfaitement légitime. Elle peut faire intervenir les hypothèses de construction H_1, "probablement" vérifiées par l'objet sous test; sous certaines conditions, elle est, elle aussi, légitime. Enfin, elle peut conduire à poser de nouvelles hypothèses H_2, dites <u>hypothèses d'optimisation</u>; nous la considérerons alors comme suspecte, et sa rentabilité devra être justifiée avec soin (cf. 5.4.2.4.).

Nous allons donner quelques exemples de telles optimisations.

6.3.1 Optimisations de degré 0

Ce type d'optimisations utilise uniquement les propriétés du H_0-contexte initial $\mathcal{C}_0 = \langle L_0, S, (\mathcal{S}_0), A \rangle$. Nous avons considéré que celui-ci réalise une représentation abstraite "<u>certainement correcte</u>" du problème du test de la propriété A sur l'objet P (ou, du moins, que tout se passe comme si elle l'était).

Ces optimisations doivent donc être considérées comme "<u>certainement légitimes</u>", en ce sens qu'elles n'altèrent certainement pas la qualité du jeu de tests obtenu lors de la phase de construction.

Elles sont ici peu nombreuses car L_0 et H_0 ont été choisis (volontairement) très restreints (cf. 6.1.2 et 6.1.5).

Un premier exemple de telles optimisations est la modification des indices des tests du jeu de tests.

Par exemple, si $\mathcal{C}_1 = \langle H_0 \sqcup H_1, (T_n^1)_{n \in \mathbb{N}} \rangle$, nous pourrons souhaiter ne considérer que certains tests T_n^1, les autres étant moins intéressants. Si nous ne souhaitons garder que les tests d'indice pair, nous prendrons

$$\mathcal{C}_2 = \langle H_0 \sqcup H_1, (T_n^2)_n \rangle$$

avec $T_n^2 = T_{2n}^1$

(cf. 5.3.1.)

Au contraire, si nous trouvons que les tests T_n^1 croissent trop vite en taille avec n, (cf. 6.2.7.4), nous serons intéressé par un étalement des indices,

$$T_n^2 = T_{E[\frac{n}{2}]}^1$$

(E[ ] est l'opération "partie entière"). Cette transformation pourra paraître spécieuse, puisqu'elle revient à dupliquer chaque test de $\mathcal{T}_1$, mais elle permet d'appliquer plus efficacement certaines méthodes décrites ci-dessous.

Un autre exemple est le réarrangement formel des tests T_n^1 sous H_0. Par exemple, nous pourrons considérer que deux expériences de T_n^1 élémentaires :

$$\text{trié}\,(F(t')) \quad , \quad \text{trié}\,(F(t''))$$

n'en forment plus qu'une en prenant leur conjonction :

$$\text{trié}\,(F(t')) \wedge \text{trié}\,(F(t'')).$$

T_n^1 peut donc être considéré comme n'étant constitué que d'une seule expérience très complexe, plutôt que d'un grand nombre de petites expériences (cf. 5.3.1 et 2.3.5).

6.3.2. Optimisations virtuellement de degré 0

Le paragraphe précédent nous a montré que les optimisations de degré 0 sont peu nombreuses et sans réel intérêt pratique, peut-être.

Ce fait tient à la petite taille de L_0 et H_0. En fait, rien ne nous aurait empêché, sinon des raisons pédagogiques évidentes, de considérer un contexte $\mathcal{C}_0$ initial plus étoffé. La correction de la représentation réalisée par celui-ci tient à la famille de structures $(\mathcal{S}_0)$ représentant les comportements éventuels du programme sous test, et non aux hypothèses formelles H_0 vérifiées par chacune des structures de $(\mathcal{S}_0)$.

Plus précisément, rien ne nous aurait empêché de prendre comme langage initial l'ensemble des symboles de L_0, mais aussi b, les $U_{n,\sigma}$, les $V_{n,\sigma,\varphi}$, les $t_{n,\sigma,\varphi}$ que nous avons introduits par la suite, ou d'autres éventuellement.

La famille de structures considérée aurait été (cf. 6.1.3) celle de toutes les structures telles que tous les symboles autres que F aient leur interprétation habituelle sur leur domaine, et telles que F soit d'interprétation calculable, et conserve, à permutation près, les tableaux.

Nous aurions pu prendre alors, comme ensemble H'_0 d'hypothèses initiales, toutes les formules du premier ordre vérifiée par toutes ces structures.

Nous pouvons noter $\mathcal{C}'_0 = \langle L'_0, S, (\mathcal{S}'_0), A \rangle$ ce contexte virtuel, qui nous aurait conduit à un jeu de tests analogue à celui effectivement construit avec $\mathcal{C}_0$. $\mathcal{C}'_0$ est un H'_0-contexte.

Notons que H'_0 contient les formules

(5), (16), (24), (33) (cf. 6.2.7.2)

Les optimisations réalisées à partir du H'_0-contexte initial virtuel $\mathcal{C}'_0$, quoique de degré 2 en toute rigueur (puisque H'_0 contient des hypothèses présentes ni dans H_0 ni dans H_1), sont tout aussi légitimes que celles de degré 0 (cf. 6.3.1.)

Le principal exemple de telles optimisations est la possibilité d'ajouter des expériences aux tests du jeu de tests considéré, et donc d'en augmenter la <u>redondance</u>

Plus précisément, soit T un ensemble d'expériences, et soit $(T_n^1)_{n \in \mathbb{N}}$ la famille de tests considérées. Alors nous avons le droit de considérer la famille $(T_n^2)_{n \in \mathbb{N}}$ telle que :

$$T_n^2 = T_n^1 \quad \text{si } n < i$$

$$T_n^2 = T_n^1 \sqcup T \quad \text{si } n \geq i$$

si T est tel que T soit formellement

démontré par $A \cup H'_0$.

Le fait de pouvoir ajouter aux tests la famille d'expériences T seulement à partir de l'indice i montre l'importance de l'étalement des indices présenté en 6.3.1.

Par exemple, si, pour une obscure raison, notre sentiment est qu'un tableau ne contenant que des 5 est particulièrement vulnérable, nous prendrons

$$T = \{ \text{trié}(F((5))),\ \text{trié}(F((5\ 5))),\ \text{trié}(F((5\ 5\ 5))) \}$$

que nous rajouterons à tous les tests d'indice supérieur ou égal à 3.

H'_0 contient les axiomes

$$\text{longueur}(\ (5)) = 1$$

et $\text{tableau}((5))$,

et donc $A \cup H'_0$ démontre $\{ \text{trié}(F((5))) \}$,

et ainsi de suite.

Par ailleurs, si nous ne sommes plus très sûr que le programme conserve les tableaux à permutation près, plutôt que de faire des expériences du type :

$$\text{trié}(F(t)),$$

nous les remplacerons toutes par des expériences du type :

$$\text{permutation}(t, F(t)) \wedge \text{trié}(F(t))$$

En effet, H'_0 contient l'axiome :

$\forall x$ permutation $(x, F(x))$,

et les deux expériences citées sont équivalentes sous H'_0 (cf. 5.3.1).

6.3.3. Optimisations de degré 1

Pour notre propos, les optimisations réellement de degré 1, c'est-à-dire fondées sur le $H_0 \cup H_1$ contexte $\mathcal{C}_1 = \langle L_1, S, (\mathcal{T}_1), A \rangle$ ne présentent guère d'intérêt.

Par contre, celles virtuellement de degré 1, c'est-à-dire utilisant éventuellement le H'_0 contexte virtuel $\mathcal{C}'_0$ (cf. 6.3.2) en plus de $\mathcal{C}_1$ sont extrêmement intéressantes.

Elles nous permettent, en effet, de <u>réduire</u> le nombre d'expériences d'un test, ou de modifier ces expériences.

Par exemple, considérons le jeu de tests $\mathcal{C}_1$ exhibé en 6.2.7.4.

T_2' est de la forme

$$\left\{ \begin{array}{l} \text{trié } (F(t_{2,\sigma_0,0})) \\ \text{trié } (F(t_{2,\sigma_0,1})) \\ \text{trié } (F(t_{2,\sigma_1,0})) \\ \text{trié } (F(t_{2,\sigma_1,1})) \end{array} \right\}$$

Mais en fait $t_{2,\sigma_0,0}$ et $t_{2,\sigma_1,0}$ sont tous deux représentants du même sous-domaine d'uniformité, celui des tableaux $(x_1\ x_2)$, avec $x_1 = x_2$.

Plus précisément, H'_0 contient l'axiome

$\forall x$ [tableau $(x) \wedge$ longueur $(x) = 2 \rightarrow$

$$\left(U_{\sigma_0}(x) \wedge V_0(x) \leftrightarrow U_{\sigma_1}(x) \wedge V_0(x) \right)]$$

Donc, par l'hypothèse d'uniformité (30)

pour $n = 2$, $\sigma = \sigma_0$, $\varphi = 0$, contenue dans H_1, et H'_0

$$\text{trié}\left(F(t_{2,\sigma_0,0})\right) \leftrightarrow \text{trié}\left(F(t_{2,\sigma_1,0})\right)$$

Plus prosaïquement, T_2' est redondant, et l'on peut sans dommage supprimer la première ou la troisième (mais non les deux!) expérience.

De même, par un raisonnement analogue, nous pouvons fixer dans T_2^1 la valeur de certaines constantes.

Par exemple, s'il est primordial d'examiner le tableau (8 17), comme H'_0 contient l'axiome :

tableau $((8\ 17)) \wedge$ longueur $((8\ 17)) = 2 \wedge$

$U_{\sigma_0}((8\ 17)) \wedge V_1((8\ 17))$,

le jeu de tests obtenu à partir de $T\,\mathcal{C}_1$ en remplaçant $t_{2,\sigma_0,1}$ par (8 17) sera tout aussi acceptable.

Les optimisations de degré 1 permettent donc, en général, de réduire notablement la taille des tests des jeux de tests, ou d'imposer à des constantes "aléatoires" des valeurs précises.

Cependant, l'hypothèse (30) (cf. 6.2.5.) n'étant que "probablement vérifiée" par le programme sous test, ces optimisations ne conduisent qu'à des résultats "probablement corrects". Aussi sont-elles moins utilisées que les optimisations de degré 0 (virtuelles).

En fait, dans la pratique, il n'est pas toujours avantageux, rentable, d'éliminer toute redondance des jeux de tests utilisés. Une certaine redondance assure en effet, sinon la correction, du moins la cohérence des hypothèses utilisées lors de la représentation abstraite de l'objet sous test.

6.3.4. Optimisations de degré 2.

Ces optimisations, à la différence des précédentes, utilisent des hypothèses supplémentaires H_2, plus fortes en général que H_0 et H_1. Ces hypothèses conduisent à effectuer une nouvelle restriction (conservative) de contexte, diminuant de ce fait la validité du principe de couplage (cf. 2.1.4), qui est à la base de la conclusion du processus de test.

Elles mettent donc en cause la consistance même du processus de test, et doivent donc être examinées avec soin.

Un exemple intéressant est le suivant. Il est raisonnable de supposer que notre programme, conçu par un cerveau humain occidental au fonctionnement sans doute proche du nôtre, est correct pour les tableaux dont tous les éléments sont égaux. Ou en tout cas, s'il ne l'est pas, c'est qu'il est "très" incorrect, et donc qu'il ne traite pas correctement un tableau de cette forme, de longueur 2.

Plus précisément, nous affirmons donc que la structure représentant réellement le comportement du programme sous test vérifie les formules :

$$[\exists x\ [\text{longueur}(x) = 2 \wedge \text{tableau}(x) \wedge U_{2,\sigma_0}(x) \wedge V_{1,\sigma_0,0}(x) \wedge \text{trié}\ (F(x))]] \rightarrow$$

$$[\forall x\ [\text{longueur}(x) = n \wedge \text{tableau}(n) \wedge U_{n,\sigma_0}(x) \wedge V_{1,\sigma_0,0}(x) \rightarrow \text{trié}\ (F(x))]]$$

pour $n \geq 2$, puisque

$$U_{n,\sigma_0}((x_1 \text{ — } x_n)) = x_1 \leq_z \cdots \leq_z x_n$$

$$V_{1,\sigma_0,0}((x_1 \text{ — } x_n)) = x_1 = \cdots = x_n .$$

Par (30) et H'₀, il suffit donc de faire l'expérience

$$\text{trié}\ (F(t_{2,\sigma_0,0}))$$

ou même

$$\text{trié}\ (F((0\ \ 0)))$$

pour pouvoir en déduire

$$\text{trié}\ (F((5\ \ 5\ \ 5)))$$

$$\text{trié}\ (F((-99\ \ -99\ \ -99\ \ -99))) \quad \text{etc..}$$

et donc supprimer toutes les expériences de cette forme des tests d'indice supérieur à 2 de $\mathcal{C}_1$.

Dans la pratique, il ne sera sans doute pas évident que l'économie réalisée par cette optimisation compense la perte de qualité du jeu de tests qu'elle occasionne.

6.3.5 Conclusion

Cette partie nous a permis de présenter la réalisation de la troisième phase du processus de test pour l'exemple que nous nous sommes proposé.

Cette phase consiste principalement dans le choix d'un test du jeu de tests élaboré lors de la phase précédente. En général, il n'existe pas de test satisfaisant, et l'on est conduit à transformer le jeu de tests considéré, à l'optimiser.

La théorie (cf. 5.3) montre que les critères guidant une telle démarche ne peuvent être qu'empiriques, subjectifs.

Nous avons décrit plusieurs de ces critères, en les classant selon la légitimité de leur utilisation.

Au degré 0, il est possible de rajouter des expériences, ou de les modifier. L'on peut en particulier augmenter la redondance du jeu de tests. Tout ceci est, sous certaines conditions, "certainement légitime".

Au degré 1, l'on peut diminuer la redondance du jeu de tests. Ceci est "probablement légitime"

Au degré 2, l'on peut, utilisant de nouvelles connaissances sur le programme, modifier profondément le jeu de tests. Ceci est, a priori, suspect, et la rentabilité de telles pratiques doit être examinée avec soin.

6.4. Conclusion

Ce chapitre avait pour but de montrer, sur un exemple concret et réaliste, l'application de la théorie développée longuement au cours des chapitres précédents.

Nous avons donc décrit, pas à pas, le processus de test de la correction d'un couple d'assertions concernant un programme de tri publié dans la littérature [Reyn 81].

Successivement, les phases de représentation, de construction, d'application ont été développée, en liaison étroite avec les notions présentées au chapitre 2, et les méthodes et outils présentés au chapitre 5.

Il est bien évident que le test finalement exhibé (cf. 6.2.7.4) n'est guère différent de celui qu'aurait probablement proposé un programmeur averti.

Certains lecteurs en auront peut-être été déçus.

Il nous semble, au contraire, que c'est là un gage de cohérence, de qualité pour notre travail que de permettre d'exprimer, de justifier, d'évaluer, par des méthodes

précisément formulées, rigoureusement fondées sur l'utilisation d'outils mathématiques largement acceptés et diffusés, des comportements empiriques ayant, d'ores et déjà, fait maintes fois leurs preuves.

C'est là l'indication que nous avons peut-être réussi, dans le cadre du développement d'une théorie mathématique abstraite, à rester fidèle à l'esprit de la notion de test, à ce qui fait sa spécificité vis-à-vis de la notion de preuve.

Chapitre 7 : Test et types abstraits algébriques

Au chapitre précédent, nous avons montré sur un exemple concret, un programme de tri, l'application de la théorie développée au cours des chapitres précédents.

Nous avons déjà eu l'occasion de signaler (cf. 3.4.3.2 et 5.2.2) que notre formalisme n'est bien adapté qu'à un type de problème très précis. Le contexte $\mathcal{C}_0$ initial doit en effet, en posant $\mathcal{C}_0 = \langle L_0, S, (\mathcal{S}_0), A \rangle$, répondre aux contraintes suivantes :

L_0 doit être dénombrable,

S doit être dénombrable,

l'interprétation des symboles de L_0 sur chacune des structures de $(\mathcal{S}_0)$ doit être calculable,

Les axiomes non-logiques de A doivent être des formules purement universelles.

Le contexte initial construit en 6.1. satisfaisait ces critères, au prix cependant d'une représentation qui a pu paraître quelque peu longue et arbitraire (cf. 6.1.4. notamment).

Le formalisme des types abstraits algébriques offre un terrain d'application privilégié à notre travail. En effet, ce formalisme est très proche de celui de la logique du premier ordre, et la phase de représentation est réduite à un simple jeu de réécritures. D'autre part, le contexte "naturellement" produit lors de cette phase satisfait généralement les conditions précédemment décrites.

Précisons ces affirmations. La donnée d'une signature est, à peu de choses près, celle d'un langage du premier ordre (au problème des types près). Un axiome d'un type abstrait est une formule du premier ordre sur le langage associé à la signature, purement universellement quantifiée. Préconditions, cas d'échecs, si_alors_sinon (cf. [Gutt 80], [Gaud 80]) peuvent s'exprimer, sous certaines conditions, par des implications.

D'autre part, les types classiques sont construits sur des signatures finies (parfois dénombrables), et, dans la mesure où l'on se restreint aux algèbres premières, l'on ne s'intéresse qu'à des algèbres dénombrables.

L'on peut se demander si cette concordance

heureuse persiste lorsque l'on considère plusieurs types concurremment. Ce problème relève de l'approche "hiérarchique" des types abstraits algébriques (cf. [Bidoit 81]). Là encore, il est facile de montrer que si les algèbres correspondant aux types inférieurs sont dénombrables, la suffisante complitude assure la dénombrabilité de celle correspondant au type d'intérêt. Comme l'on ne s'intéresse en général qu'à des extensions suffisamment complètes, les conditions d'application de notre formalisme sont réunies dans ce cas.

Il est intéressant de noter que la forme des axiomes d'un type abstrait répond à des exigences extérieures à notre propos, mais conduisant aux mêmes résultats. En effet, il a été montré [CIP 80] que l'introduction de quantificateurs existentiels empêche, en général, l'existence d'une algèbre initiale, comme le montre l'exemple suivant.

<u>Type</u> : Exemple

<u>Operations</u>

f : Exemple $\rightarrow$ Exemple

a : $\rightarrow$ Exemple

<u>Axiomes</u>

$\exists x \quad f(x) = a$

<u>Fin</u>

Nous allons, dans ce chapitre, à l'aide d'un exemple simple, montrer l'application de notre formalisme au test d'un axiome sur une algèbre.

Lorsque les méthodes employées, lors de la phase de construction et lors de l'optimisation notamment seront analogues à celles exposées en détail au chapitre 6, nous nous permettrons d'être plus bref, et d'indiquer au lecteur, par une référence, l'endroit où il trouvera des justifications plus détaillées. Ceci nous permettra d'insister sur les points spécifiques de l'utilisation des types abstraits algébriques.

7.1 Position du problème et représentation

Considérons la présentation hiérarchique suivante File (Entier) [Gaud 79]

Type d'intérêt File ; Entier

Opérations

ajouter : Entier × File → File

enlever : File → File

file_vide : → File

premier : File → Entier

Axiomes

$f \in$ File, $n \in$ Entier

① enlever (ajouter (n, f)) = si f = file_vide ()
alors file_vide ()
sinon ajouter (n, enlever (f))

② enlever (file_vide ()) = file_vide ()

③ premier (ajouter (n, f)) = si f = file_vide
alors n
sinon premier (f)

Préconditions

$f \in$ File

premier (f) : ¬ (f = file_vide)

Fin

<u>Type inférieur</u> : Entier

<u>Opérations</u>

0 : $\longrightarrow$ Entier

succ : Entier $\longrightarrow$ Entier

pred : Entier $\longrightarrow$ Entier

<u>Axiomes</u>

$n \in$ Entier

① succ (pred (n)) = n

② pred (succ (n)) = n

<u>Fin</u>

Nous avons légèrement modifié la présentation donnée en [Gaud 79], notamment par la suppression du type inférieur Bool. Celui-ci ne servait, en effet, qu'à faire de si_alors_sinon une opération à part entière, de type Bool × File × File → File, ou Bool × Entier × Entier → Entier. Nous préférons laisser à ce symbole son statut de "méta-opération".

7.1.1 Position du problème

Considérons une File (Entier)-algèbre X. Conformément aux habitudes, nous notons

X_{File} , X_{Entier}

les ensembles supports des types File et Entier décrits dans la présentation hiérarchique ci-dessus.

Puisque X est une File (Entier)-algèbre, nous avons sur X des applications associées à chacun des symboles de la présentation. Nous les notons :

$ajouter_X : X_{Entier} \times X_{File} \rightarrow X_{File}$

etc.

Conformément à l'esprit de l'approche hiérarchique des types abstraits algébriques, le problème qui se pose est le suivant:

Supposons que X valide les axiomes des types inférieurs.

Peut-on déterminer si X valide les axiomes du type d'intérêt ?

Plus précisément, en notant

E_{File} , E_{Entier}

les ensembles d'axiomes des types File et Entier, le problème s'énonce ainsi :

Supposons que X soit un File (Entier), E_{Entier}-algèbre.

Peut-on déterminer si X est une File(Entier), $E_{Entier} \cup E_{File}$-algèbre?

Pour cela, il suffit bien évidemment de vérifier que X vérifie chacun des axiomes ①, ②, ③ du type File, compte tenu de la précondition sur "premier".

Le problème que nous allons résoudre est donc le suivant.

<u>Supposons que X est une File(Entier), E_{Entier}-algèbre. X vérifie-t-il l'axiome ④ du type File (compte tenu de la précondition !)?</u>

Conformément à l'introduction de ce chapitre, nous supposons que X est une File(Entier)-algèbre <u>première</u>, et que l'extension de Entier à File(Entier) réalisée par X est <u>suffisamment complète</u>.

La seconde condition revient à imposer que X considérée seulement comme Entier-algèbre par le foncteur d'oubli est une Entier-algèbre première

Intuitivement ces conditions signifient que l'on "sait" a priori que X est une Entier-algèbre <u>raisonnable</u>, et que l'extension a, elle aussi, été réalisée de manière <u>raisonnable</u>.

7.1.2. Langage et ensemble de base : L_0, S

Nous prenons, conformément à la méthode proposée en 6.1.1.,

$$S = X_{File} \sqcup X_{Ent} \sqcup \{\perp\}. \qquad (1)$$

Nous prenons comme langage initial celui déduit des signatures de la présentation, auquel nous ajouter les symboles de types.

$$L_0 = \{ \text{File}, \text{Entier}, \perp \qquad (2)$$
$$\text{ajouter}, \text{enlever}, \text{premier}, \text{file\_vide}$$
$$\text{succ}, \text{pred}, 0 \}$$

Par primarité, X_{File} et X_{Ent} sont dénombrables. S est donc dénombrable et L_0 l'est aussi.

7.1.3 Structures : $(\mathcal{S}_0)$

Nous construisons la famille $(\mathcal{S}_0)$ de manière analogue à celle proposée en 6.1.3. La famille $(\mathcal{S}_0)$ est grossièrement celle de toutes les File(Entier)-algèbres vérifiant les conditions présentées en 7.1.1., et telles que tous les symboles soient d'interprétation calculable.

Plus précisément, $(\mathcal{S}_0)$ est la famille de toutes les $L_0(S)$-structures $\mathcal{S}_0$ telles que les conditions suivantes soient vérifiées.

- $File_{\mathcal{S}_0}(x)$ est vrai si et seulement si $x \in X_{File}$. Idem pour $Entier_{\mathcal{S}_0}$ et $\perp_{\mathcal{S}_0}$ avec X_{Entier} et $\{\perp\}$.
- $0_{\mathcal{S}_0}$ est une constante de X_{Entier}
- $succ_{\mathcal{S}_0}$ et $pred_{\mathcal{S}_0}$ sont des applications de X_{Entier} dans X_{Entier}, et valent $\perp$ ailleurs.
- $premier_{\mathcal{S}_0}$ est une application de X_{File} dans X_{Entier}, et vaut $\perp$ ailleurs.
- $enlever_{\mathcal{S}_0}$ est une application de X_{File} dans X_{File}, et vaut $\perp$ ailleurs
- $ajouter_{\mathcal{S}_0}$ est une application de $X_{Entier} \times X_{File}$ dans X_{File}, et vaut $\perp$ ailleurs.
- $file\_vide_{\mathcal{S}_0}$ est une constante de X_{File}

De plus, nous imposons à tous les symboles d'être d'interprétation <u>calculable</u> sur $\mathcal{S}_0$, et de vérifier les conditions suivantes

- X_{Entier} est <u>engendré</u> par
  $0_{\mathcal{S}_0}$, $pred_{\mathcal{S}_0}$, $succ_{\mathcal{S}_0}$ (3)

- X_{File} est <u>engendré</u> par
  $file\_vide_{\mathcal{S}_0}$, $ajouter_{\mathcal{S}_0}$, $enlever_{\mathcal{S}_0}$ (4)
  et les éléments de X_{Entier}

- $\mathcal{L}$ valide les formules suivantes

$$\forall n \, [Entier(n) \rightarrow succ(pred(n)) = n] \quad (5)$$

$$\forall n \, [Entier(n) \rightarrow pred(succ(n)) = n]. \quad (6)$$

Ces conditions sont directements déduites de la discussion présentée en 7.1.1. concernant la structure "raisonnable" de X

7.1.4. Propriété sous test : A

Ce point est sans doute le plus délicat de cette partie triviale par ailleurs.

Le problème réside dans l'interprétation sémantique du si_alors_sinon et des préconditions (et éventuellement des cas d'échec) présents dans un type abstrait.

Jusqu'ici, personne, à notre connaissance, n'a défini précisément une sémantique de ces objets réalisant un large consensus dans la communauté scientifique.

Le cœur du problème est la "nature" du symbole "=" dans la condition du si_alors_sinon et dans la précondition du type File.

Suivant, semble-t-il, la voie tracée par [CIP 80], nous considérerons que tous les symboles "=" ont la même valeur dans l'axiome ② du type File que nous testons sur X, aussi bien que dans la précondition sur "premier", et que cet "=" est celui de la logique du premier ordre

Nous procédons alors de la manière suivante.

L'axiome considéré est :

enlever (ajouter (n, f)) = si f = file_vide () (7)
alors file_vide ()
sinon ajouter (n, enlever (f))

Divisons-le en deux axiomes munis de préconditions (cf. [Bido 81]) :

f = file_vide () : enlever (ajouter (n, f)) = file_vide ()
(8)

¬(f = file_vide ()) :
enlever (ajouter (n, f)) = ajouter (n, enlever (f)) (9)

Rajoutons éventuellement les préconditions résultant de l'apparition du symbole "premier" dans chacun de ces axiomes (il n'apparaît nulle part ici).

Transcrivons les préconditions sous forme d'implication :

f = file_vide () → enlever (ajouter (n, f)) = file_vide () (10)

¬f = file_vide () →
enlever (ajouter (n, f)) = ajouter (n, enlever (f)) (11)

Réunissons par une conjonction ces sous-axiomes et fermons le tout.

$$
\begin{array}{l}
\forall f \quad [\text{File}(f) \rightarrow \\
\quad \forall n\, [\text{Entier}(n) \rightarrow \\
\qquad [(f = \text{file\_vide} \rightarrow \\
\qquad\quad \text{enlever}(\text{ajouter}(n, f)) = \text{file\_vide}()) \wedge \\
\qquad (\neg(f = \text{file\_vide}) \rightarrow \\
\qquad\quad \text{enlever}(\text{ajouter}(n, f)) = \\
\qquad\qquad \text{ajouter}(n, \text{enlever}(f)))]]]
\end{array}
\tag{12}
$$

A est la $L_0(S)$-théorie ayant (12) pour seul axiome non-logique. C'est une formule purement universelle. A n'ayant qu'un seul axiome non-logique, nous sommes dans les conditions d'application du théorème fondamental du test (cf. 4.4.3)

A titre informatif, montrons quelle aurait été, selon la méthode exposée plus haut, la traduction de l'axiome ③ du type File.

Nous aurions divisé l'axiome en deux parties :

$f = \text{file\_vide}() \rightarrow \text{premier}(\text{ajouter}(n, f)) = n$

$\neg f = \text{file\_vide}() \rightarrow \text{premier}(\text{ajouter}(n, f)) = \text{premier}(f)$.

Puis nous aurions fait intervenir les préconditions. Ceci nous aurait conduit aux formules suivantes :

$(f = file\_vide()) \wedge \qquad \neg(ajouter(n,f) = file\_vide())$
$\quad \rightarrow premier(ajouter(n,f)) = n$

$\neg(f = file\_vide()) \wedge \qquad \neg(ajouter(n,f) = file\_vide())$
$\quad \wedge \neg(f = file\_vide())$
$\quad \rightarrow premier(ajouter(n,f)) = premier(f)$

D'où la formule suivante :

$\forall f\ [File(f) \rightarrow$
$\quad \forall n\ [Entier(n) \rightarrow$
$\quad [\ [\ f = file\_vide() \wedge$
$\qquad \neg\ ajouter(n,f) = file\_vide()$
$\qquad \rightarrow premier(ajouter(n,f)) = n\ ] \wedge$
$\quad [\ \neg\ f = file\_vide() \wedge$
$\qquad \neg\ ajouter(n,f) = file\_vide() \wedge$
$\qquad \neg\ f = file\_vide()$
$\qquad \rightarrow premier(ajouter(n,f)) = premier(f)\ ]]]]$.

Ces transformations sont donc tout à fait cohérentes avec la notion <u>intuitive</u> de si_alors_sinon et de précondition, même si leur fondement théorique reste encore flou.

7.1.5. Hypothèses initiales : H_0

Conformément aux remarques énoncées aux paragraphes 6.1.3 et 6.3.2., rien ne nous empêche de prendre pour ensemble d'hypothèses initiales l'ensemble de <u>toutes</u> les $L_0(S)$-formules valides sur chacune des structures de la famille $(\mathcal{S}_0)$.

Cet ensemble contient donc, en particulier, les formules suivantes :

$$\forall n \; [\, \text{Entier}(n) \rightarrow \text{succ}(\text{pred}(n)) = n \,] \qquad (5)$$

$$\forall n \; [\, \text{Entier}(n) \rightarrow \text{pred}(\text{succ}(n)) = n \,]. \qquad (6)$$

H_0 est la $L_0(S)$-théorie ayant l'ensemble de ces hypothèses initiales pour ensemble d'axiomes non-logiques.

H_0 est donc une "énorme" théorie, et nous serons loin d'utiliser tous ses axiomes. Par 5.3.2., il nous sera de toute façon possible, a posteriori, de nous restreindre aux seuls axiomes que nous utilisons réellement.

7.1.6 Conclusion

Nous avons donc défini un H_0-contexte initial $\mathcal{C}_0 = \langle L_0, S, (\mathcal{S}_0), A \rangle$ représentant le problème concret du test de l'axiome ① du type File sur la File(Entier)-algèbre X.

La simplicité de cette représentation, son aspect intuitif contrastent agréablement avec le travail accompli dans la partie 6.1 qui a pu, parfois, sembler arbitraire et obscur. Ici, le langage L_0, l'ensemble de base S, la famille de $L_0(S)$-structure $(\mathcal{S}_0)$ et la théorie d'hypothèses H_0 dérivent directement et mécaniquement de la donnée de la présentation (hiérarchique) File(Entier), et de la File(Entier)-algèbre X.

L'élaboration de la théorie sous test, A, est rendue plus complexe par l'absence de sémantique rigoureuse pour le si_alors_sinon et les préconditions intervenant éventuellement dans l'axiome sous test. Nous avons cependant décrit une méthode mécanique permettant une élaboration conforme à l'intuition (cf. 7.1.4).

Remarquons que la hiérarchie introduite dans la présentation File(Entier) pour un

grand rôle lors de la phase de représentation.

Nous considérons en effet que les axiomes des types inférieurs ont déjà été validés sur l'algèbre considérée, et que celle-ci est une "bonne" algèbre (une algèbre première) vis à vis de ces types.

Ceci est conforme à l'esprit précisé par [Prido 81] faisant des types inférieurs des types prédéfinis, figés, et du type d'intérêt un type susceptible de modifications et d'améliorations.

L'hypothèse de primarité sur X (en tant que File (Entier)-algèbre) peut être vue comme l'analogue de l'hypothèse formulée en 6.1.3. énonçant que le programme I s'arrête en fournissant un résultat pour toute valeur d'entrée. L'important n'est pas qu'elle soit réellement vérifiée, mais que tout se passe comme si elle l'était.

Remarquons enfin que le contexte $\mathcal{C}_0$ vérifie "naturellement" les propriétés énoncées au début de ce chapitre.

7.2. Phase de construction

Cette partie est strictement analogue à la partie 6.2.

Etant donné le H_0-contexte $\mathcal{C}_0 = \langle L_0, S, (\mathcal{S}_0), A \rangle$, il nous faut construire un $H_0 \sqcup H_1$-contexte $\mathcal{C}_1 = \langle L_1, S, (\mathcal{S}_1), A \rangle$, restriction conservative de $\mathcal{C}_0$, admettant un jeu de tests acceptable $\mathcal{T} = \langle H_0 \sqcup H_1, (T_n)_{n \in \mathbb{N}} \rangle$. De plus, la famille $(\mathcal{S}_1)$ doit être telle que le principe de couplage (cf. 2.1.4) soit applicable. Les hypothèses H_1 doivent donc être "raisonnables", "probablement" vérifiées par la structure $\mathcal{S}_0$ représentant réellement le comportement de la File (Entier)-algèbre X.

Puisque notre construction ne dépend que du H_0-contexte $\mathcal{C}_0$, le fait que nous travaillions dans le cadre des types abstraits algébriques n'a, en toute rigueur, plus d'importance, et la méthode employée en 6.2. est applicable.

Cependant, notre connaissance extérieure du problème sera cruciale pour justifier les hypothèses d'uniformité et de régularité que nous utiliserons. Mais de telles justifications sont extérieures au formalisme que nous utilisons, et relèvent de la notion de qualité intrinsèque

Ces justifications seront fondées sur les remarques suivantes. (cf. 7.1.3.)

Soit n un individu tel que Entier(n). Alors $n \in X_{Entier}$, et, par hypothèse, pour toute structure $\mathcal{S}_0$, il existe t tel que :

$$n = t_{\mathcal{S}_0} \qquad (13)$$

où t est un terme, <u>bien formé en tant que terme abstrait de File(Entier)</u>, ne contenant que les symboles 0, pred et succ. Par les axiomes (5) et (6) (cf. 7.1.5.), on peut même prendre $t = 0$, $t = succ(succ \ldots (succ(0)) \ldots)$ ou $t = pred(pred(\ldots (pred(0)) \ldots)$. Nous noterons, par abus de langage, $t = 0$, $t = succ^i(0)$, $t = pred^j(0)$.

Soit f un individu tel que File(f). Alors $f \in X_{File}$, et, par hypothèse, pour toute structure $\mathcal{S}_0$, il existe un terme $t[x_1 \ldots x_i]$, <u>bien formé en tant que terme du type File construit sur les variables $x_1 \ldots x_i$ de type Entier, et ne contenant que les opérations file_vide, enlever, ajouter</u>, tel que $f = t_{\mathcal{S}_0}[n_1, \ldots, n_i]$ (14)

$n_1, \ldots, n_i$ étant des individus de X_{Entier}.

Comme en 6.2, nous partons avec

$$L_1 := L_0 \ , \quad H_1 := \emptyset \ , \quad (\mathcal{S}_1) := (\mathcal{S}_0)$$

7.2.1. Première réduction

Nous cherchons à tester l'ensemble de propriétés

$$
\begin{aligned}
A = \{ & \forall f\, [\mathrm{File}(f) \rightarrow \\
& \quad \forall n\, [\mathrm{Entier}(n) \rightarrow \\
& \quad [\, [f = \mathrm{file\_vide}() \rightarrow \\
& \qquad \mathrm{enlever}(\mathrm{ajouter}(n, f)) = \mathrm{file\_vide}\,] \wedge \\
& \quad [\neg f = \mathrm{file\_vide}() \rightarrow \\
& \qquad \mathrm{enlever}(\mathrm{ajouter}(n, f)) = \mathrm{ajouter}(n, \mathrm{enlever}(f))]]]]\}.
\end{aligned}
\tag{12}
$$

Cette propriété est formellement équivalente à l'ensemble de propriétés :

$$B \sqcup C$$

avec

$$
\begin{aligned}
B = \{ & \forall f\, [\mathrm{File}(f) \rightarrow \\
& \quad \forall n\, [\mathrm{Entier}(n) \rightarrow \\
& \quad [f = \mathrm{file\_vide}() \rightarrow \\
& \qquad \mathrm{enlever}(\mathrm{ajouter}(n, f)) = \mathrm{file\_vide}()]]]\}
\end{aligned}
\tag{15}
$$

et

$$
\begin{aligned}
C = \{ & \forall f\, [\mathrm{File}(f) \rightarrow \\
& \quad \forall n\, [\mathrm{Entier}(n) \rightarrow \\
& \quad [\neg f = \mathrm{file\_vide}() \rightarrow \\
& \qquad \mathrm{enlever}(\mathrm{ajouter}(n, f)) = \mathrm{ajouter}(n, \mathrm{enlever}(f))]]]
\end{aligned}
\tag{16}
$$

Par le théorème de localisation (cas fini) (cf. 5.2.3) nous sommes ramené à rechercher des jeux de tests acceptables pour les $H_0 \cup H_1$ - contextes

$$\langle L_1, S_1, (\mathcal{P}_1), B \rangle$$

et $$\langle L_1, S_1, (\mathcal{P}_1), C \rangle$$

7.2.2. Etude de B : stratification

Nous sommes conduit à rechercher un jeu de test acceptable sur le contexte :

$$< L_1, S_1, (\mathcal{S}_1), B >$$

avec

$$B = \{ \forall f \, [\text{File}(f) \rightarrow \forall n \, [\text{Entier}(n) \rightarrow [\, f = \text{file\_vide}() \rightarrow \text{enlever}(\text{ajouter}(n, f)) = \text{file\_vide}() \,]]] \} \qquad (15)$$

B est formellement équivalente sous H_0 à

$$\{ \forall n \, [\text{Entier}(n) \rightarrow \text{enlever}(\text{ajouter}(n, \text{file\_vide}())) = \text{file\_vide}()] \qquad (17)$$

que nous notons aussi B.

En effet, par 7.1.3 et 7.1.5, H_0 contient l'axiome

$$\text{File}(\text{file\_vide}()) . \qquad (18)$$

Conformément à la méthode de 6.2.3, 6.2.4 et 6.2.5, nous allons stratifier les n tels que Entier(n) pour poser des hypothèses d'uniformité raisonnables.

Ajoutons au langage les symboles U_0, U_+, U_-

$$L_1 = L_1 \sqcup \{ U_0, U_+, U_- \} \qquad (19).$$

Etendons les structures de la famille $(\mathcal{S}_1)$ de telle manière que, pour toute structure $\mathcal{S}_1$ de $(\mathcal{S}_1)$,

$U_0 \mathcal{S}_1(x)$ est vrai si et seulement si

$x \in X_{Entier}$, et $x = 0_{\mathcal{S}_1}$,

$U_+ \mathcal{S}_1(x)$ est vrai si et seulement si

$x \in X_{Entier}$ et $x = succ^i_{\mathcal{S}_1}(0_{\mathcal{S}_1})$ avec $i > 0$

$U_- \mathcal{S}_1(x)$ est vrai si et seulement si

$x \in X_{Entier}$ et $x = pred^j_{\mathcal{S}_1}(0_{\mathcal{S}_1})$ avec $j > 0$

Par les remarques de l'introduction de la partie 7.2., la formule :

$$\forall n \, [\, Entier(n) \leftrightarrow U_0(n) \vee U_+(n) \vee U_-(n) \,] \qquad (20)$$

est valide pour toutes les structures $\mathcal{S}_1$ de $(\mathcal{S}_1)$.

Sans rien changer, nous pouvons donc étendre H_1 avec $\{(20)\}$:

$$H_1 := H_1 \sqcup \{(20)\},$$

et avec toutes les formules valides sur chacune des structures $\mathcal{S}_1$ de $(\mathcal{S}_1)$ faisant intervenir ces nouveaux symboles.

Nous considérerons en particulier les axiomes :

$$\begin{aligned} &U_0(0) \\ &U_+(succ(0)) \\ &U_-(pred(0)) \\ &\text{etc.} \end{aligned} \qquad (21)$$

Par (20), B est équivalent à l'ensemble de formules suivant.

$$B_0 \sqcup B_+ \sqcup B_-$$

avec

$$B_0 = \{ \forall n \, [U_0(n) \rightarrow \text{enlever}(\text{ajouter}(n, \text{file\_vide}())) = \text{file\_vide}()] \} \quad (22)$$

$$B_+ = \{ \forall n \, [U_+(n) \rightarrow \text{enlever}(\text{ajouter}(n, \text{file\_vide}())) = \text{file\_vide}()] \} \quad (23)$$

$$B_- = \{ \forall n \, [U_-(n) \rightarrow \text{enlever}(\text{ajouter}(n, \text{file\_vide}())) = \text{file\_vide}()] \} \quad (24)$$

Par le théorème de localisation (cas fini), nous sommes conduit à construire des jeux de tests acceptables sur les

$< L_1, S_1 (\mathcal{S}_1), B_0 >$

$< L_1, S_1 (\mathcal{S}_1), B_+ >$

$< L_1, S_1 (\mathcal{S}_1), B_- >$.

7.2.3. Etude de B : hypothèse d'uniformité et jeux de tests

Examinons tout d'abord le contexte :

$$\langle L_1, S, (\mathcal{S}_1), B_0 \rangle.$$

Parmi les axiomes (21) se trouve l'axiome :

$$\forall n [\, U_0(n) \leftrightarrow n = 0 \,] \qquad (25)$$

Donc sous H_1, B_0 est equivalent à

$$\{ \text{enlever}(\text{ajouter}(0, \text{file\_vide}())) = \text{file\_vide}() \} \qquad (26)$$

qui est effectivement vérifiable sur ce contexte.

Un jeu de tests acceptable est donc donné par $\langle H_0 \sqcup H_1, (T_i)_{i \in \mathbb{N}^-} \rangle$

avec $T_i = \{ (26) \} \quad \forall i \in \mathbb{N}^-$.

Examinons maintenant les $H_0 \sqcup H_1$-contextes :

$$\langle L_1, S, (\mathcal{S}_1), B_+ \rangle$$

$$\langle L_1, S, (\mathcal{S}_1), B_- \rangle.$$

Entier étant un type inférieur, nous avons supposé que X est une "bonne" Entier-algèbre. En particulier, il est raisonnable de penser que les sous-domaines U_+ et U_- sont <u>uniformes</u> vis-à-vis de B_+ et B_- (nous nous permettant ici un léger abus de langage).

Nous augmentons donc H_1 des hypothèses suivantes :

$$\exists n\,[\,U_+(n) \wedge \text{enlever}(\text{ajouter}(n, \text{file\_vide}())) = \text{file\_vide}()\,] \rightarrow \forall n\,[\,U_+(n) \rightarrow \text{enlever}(\text{ajouter}(n, \text{file\_vide}())) = \text{file\_vide}()\,] \qquad (27)$$

$$\exists n\,[\,U_-(n) \wedge \text{enlever}(\text{ajouter}(n, \text{file\_vide}())) = \text{file\_vide}()\,] \rightarrow \forall n\,[\,U_-(n) \rightarrow \text{enlever}(\text{ajouter}(n, \text{file\_vide}())) = \text{file\_vide}()\,] \qquad (28)$$

(cf. 6.2.5)

$$H_1 := H_1 \sqcup \{(27), (28)\}$$
$$(\mathcal{S}_1) := \{\, \mathcal{S}_1 \in (\mathcal{S}_1),\ \mathcal{S}_1 \models (27) \wedge (28) \,\}$$

Nous restreignons $(\mathcal{S}_1)$ en conséquence.

Suivant 6.2.6, nous rajoutons à L_1 des symboles u_+, u_-.

$$L_1 = L_1 \sqcup \{u_+, u_-\}$$

Nous étendons les structures $\mathcal{S}_1$ de $(\mathcal{S}_1)$ de toutes les manières possibles telles que :

$u_+\mathcal{S}_1 = \text{succ}^i_{\mathcal{S}_1}(0_{\mathcal{S}_1})$ pour un certain $i > 0$,

et $u_-\mathcal{S}_1 = \text{pred}^j_{\mathcal{S}_1}(0_{\mathcal{S}_1})$ pour un certain $j > 0$.

Nous rajoutons les axiomes :

$$U_+(u_+) \qquad (29)$$

$$U_-(u_-) \qquad (30)$$

$$H_1 := H_1 \sqcup \{(29), (30)\}$$

Nous obtenons pour le contexte $\langle L_1, S, (\mathcal{S}_1), B_+ \rangle$ le jeu de tests acceptable

$$\langle H_0 \sqcup H_1, (T_i)_{i \in \mathbb{N}} \rangle$$

avec $T_i = \{ \text{enlever}(\text{ajouter}(u_+, \text{file\_vide}())) = \text{file\_vide}() \} \quad \forall i \in \mathbb{N}$.

Nous obtenons pour le $(H_0 \sqcup H_1, -)$contexte $\langle L_1, S, (\mathcal{S}_1), B_- \rangle$ le jeu de tests acceptable

$$\langle H_0 \cup H_1, (T_i)_{i \in \mathbb{N}} \rangle$$

avec $T_i = \{ \text{enlever}(\text{ajouter}(u_-, \text{file\_vide}())) = \text{file\_vide}() \} \quad \forall i \in \mathbb{N}$.

En effet, comme (26), ces formules sont effectivement vérifiables sur ces contextes.

7.2.4. Etude de C : hypothèse de régularité

Nous sommes conduit, par 7.2.1., à étudier l'existence d'un jeu de tests acceptable sur le $H_0 \sqcup H_1$ - contexte

$$\langle L_1, S_1(\mathcal{S}_1), C \rangle$$

avec

$$C = \{ \forall f\ [\mathrm{File}(f) \rightarrow \forall n\ [\mathrm{Entier}(n) \rightarrow [\neg\ f = \mathrm{file\_vide}() \rightarrow \mathrm{enlever}(\mathrm{ajouter}(n, f)) = \mathrm{ajouter}(n, \mathrm{enlever}(f))]]]\} \qquad (16)$$

Pour clarifier l'exposé, selon 6.2.2., introduisons le nouveau symbole c, avec la formule.

$$\forall f\ [\ c(f) \leftrightarrow \forall n\ [\mathrm{Entier}(n) \rightarrow [\neg\ f = \mathrm{file\_vide}() \rightarrow \mathrm{enlever}(\mathrm{ajouter}(n, f)) = \mathrm{ajouter}(n, \mathrm{enlever}(f))]]] \qquad (31)$$

$$L_1 := L_1 \cup \{c\}$$

$$H_1 := H_1 \sqcup \{(31)\}$$

Nous étendons chacune des structures de $(\mathcal{S}_1)$ par la définition (31).

C est donc, sous H_1, équivalent à

$$\{ \forall f\ [\mathrm{File}(f) \rightarrow c(f)]\}$$

que nous notons encore C (cf. 6.2.2.)

Selon la remarque proposée en introduction à la partie 7.2., étant donné une structure $\mathcal{S}_1$ de $(\mathcal{S}_1)$ et un individu f tel que File (f), il existe un terme, $t[x_1 \dots x_{p_t}]$, bien formé en tant que terme abstrait du type File, ne faisant intervenir que les symboles : enlever, ajouter et file_vide, construit sur les variables de type Entier $x_1 - x_p$, tel que

$$f = t_{\mathcal{S}_1}[n_1 \dots n_p],$$

$n_1 \dots n_p$ étant des <u>individus</u> de X_{Entier}.

La hauteur de t est sa hauteur en tant que terme abstrait, par rapport aux symboles de File.

file_vide est de hauteur 1,

ajouter (n_1, enlever (ajouter (n_2, file_vide()))) est de hauteur 4

Nous allons stratifier sur la hauteur des termes de cette forme.

Par la remarque précédente, si, pour une structure $\mathcal{S}_1$ de $(\mathcal{S}_1)$,

$$\mathcal{S}_1 \models \forall n_1 \dots \forall n_{p_t} \; [\text{Entier}(n_1) \wedge \dots \wedge \text{Entier}(n_{p_t}) \rightarrow c(t[n_1 \dots n_{p_t}])]$$

pour tout terme t de la forme indiquée, alors

$$\mathcal{S}_1 \models \forall f \; [\text{File}(f) \rightarrow c(f)]$$

Pour chaque terme t de la forme considérée, construit sur les variables $x_1 \dots x_{p_t}$, introduisons le symbole d_t

$$L_1 := L_1 \sqcup \{ dt, t \text{ terme de la forme sus-dite} \}$$

avec la définition

$$dt \leftrightarrow \forall n_1 \dots \forall n_{p_t} \; [\text{Entier}(n_1) \wedge \dots \wedge \text{Entier}(n_{p_t}) \rightarrow c(t[n_1 \dots n_{p_t}])] \qquad (32)$$

Etendons les structures de $(\mathcal{S}_1)$ par ces définitions, et augmentons H_1 en conséquence.

$$H_1 := H_1 \sqcup \{ (32), t \text{ variant sur les termes précédents} \}$$

La remarque précédente s'écrit ainsi :

si $\mathcal{S}_1 \models d_t$ pour tout t, alors $\mathcal{S}_1 \models C$ (33)

Comme en 6.2.2, considérons l'hypothèse de régularité suivante,

$$\bigwedge_{\text{hauteur}(t) \leq k} d_t \rightarrow C. \qquad (34)$$

Cette hypothèse est justifiée par la remarque (33) qui affirme que si $k \simeq \infty$, alors (34) est vérifiée par toute structure de $(\mathcal{S}_1)$.

Prenons donc k grand, rajoutons (34) à H_1 et réduisons $(\mathcal{S}_1)$ en conséquence

$$H_1 := H_1 \sqcup \{(34)\}$$

Sous H_1, C est équivalent à

$$\bigsqcup_{i=1}^{\infty} \left\{ \bigwedge_{\text{hauteur}(t)=i} dt \right\} \qquad (35)$$

Par le théorème de localisation, nous sommes donc ramené au test de

$$D_i = \left\{ \bigwedge_{\text{hauteur}(t)=i} dt \right\} \qquad (36)$$

i étant considéré comme une constante.

7.2.5 Etude de D

Nous sommes donc ramené à la recherche d'un jeu de tests acceptable sur le $H_0 \cup H_1$-contexte :

$$< L_1, S_1, (\mathcal{S}_1), D >$$

avec

$$D = \{ \bigwedge_{\text{hauteur}(t)=i} d_t \} ,$$

i étant considéré comme une constante supérieure ou égale à 1

Mais D est formellement équivalent à

$$\bigsqcup_{\text{hauteur}(t)=i} \{ d_t \} \qquad (38)$$

et par le théorème de localisation (cas fini, puisqu'il n'y a qu'un nombre fini de tels t), nous sommes ramené au cas de :

$$E_t = \{ d_t \} . \qquad (39)$$

7.2.6. Etude de E

Nous cherchons donc un jeu de tests acceptable sur le $H_0 \sqcup H_1$ - contexte

$$\langle L_1, S_1, (\mathcal{S}_1), E \rangle$$

avec

$$E = \{ dt \}$$

t étant un terme de forme adéquate, fixé, de hauteur i, construit sur $n_1 \dots n_p$

Pour alléger les écritures, introduisons le symbole e avec la définition suivante :

$$\forall f \forall n \; [\, e(f, n) \leftrightarrow$$
$$[\, \neg\, f = \text{file\_vide}() \rightarrow \qquad (40)$$
$$\text{enlever}(\text{ajouter}(n, f)) = \text{ajouter}(n, \text{enlever}(f))]]\,.$$

Ajoutons alors e à L_1, étendons les structures de $(\mathcal{S}_1)$ par (40) et ajoutons (40) à H_1

$$L_1 := L_1 \sqcup \{e\}$$
$$H_1 := H_1 \sqcup \{(40)\}\,.$$

Sous ces hypothèses, E est formellement équivalent à :

$$\{\, \forall n_1 \dots \forall n_p \forall n_{p+1}$$
$$[\text{Entier}(n_1) \wedge \dots \wedge \text{Entier}(n_p) \wedge \text{Entier}(n_{p+1}) \rightarrow \qquad (41)$$
$$e(\; t[n_1 \dots n_p], n_{p+1}) \;]\,\}$$

que nous notons encore E.

Il n'y a aucune raison de séparer, dans cette formule, les n_i de n_{p+1}. Un examen attentif de la forme de t montre que leurs rôles sont analogues.

Remarquons d'autre part que les n_i et n_{p+1} étant fixés, la formule :

$$e\,(t[n_1 \ldots n_p], n_{p+1})$$

est effectivement vérifiable sur $(\mathcal{S}_1)$.

Le lecteur assidu, averti et patient aura sans doute deviné comment produire un jeu de tests acceptable pour E sur un contexte restreint à partir du précédent par l'introduction d'hypothèses d'uniformité et de nouveaux symboles, comme en 7.2.2 et 7.2.3.

Il nous faut, en fait, appliquer successivement $p+1$ fois la méthode décrite dans ces paragraphes pour éliminer progressivement tous les quantificateurs de E.

Une telle description serait longue, fastidieuse et sans intérêt réel pour le lecteur. Aussi nous contentons-nous d'en esquisser le résultat.

Nous allons introduire successivement $3 \times (p+1)$ nouveaux symboles, pour chaque indice i, $\sigma_{i,0}$, $\sigma_{i,+}$, $\sigma_{i,-}$, tels que

$U_0(\sigma_{i,0})$, $U_+(\sigma_{i,+})$, $U_-(\sigma_{i,-})$ (cf. 7.2.2)

étendant les structures de $(\mathcal{S}_1)$ de toutes les manières possibles compatibles avec ces restrictions, et rajoutant ces formules comme axiomes de H_1.

Nous restreindrons ensuite les structures de $(\mathcal{S}_1)$ par l'introduction d' hypothèses d'uniformité. Celles de rang i seront par exemple l'ensemble de formules du type $(1 \leq i \leq p+1)$

(42)

$$\exists n_i [\ U(n_i) \wedge$$
$$\forall n_{i+1} \ldots \forall n_p \forall n\ [\ \mathrm{Entier}(n_{i+1}) \wedge \ldots \wedge \mathrm{Entier}(n_p) \wedge \mathrm{Entier}(n_{p+1}) \rightarrow$$
$$e(t[\alpha_1, \ldots, \alpha_{i-1}, n_i, n_{i+1} \ldots n_p], n_{p+1})\ ]\ ] \rightarrow$$
$$\forall n_i\ [\ U(n_i) \rightarrow$$
$$\forall n_{i+1} \ldots \forall n_p \forall n\ [\mathrm{Entier}(n_{i+1}) \wedge \ldots \wedge \mathrm{Entier}(n_p) \wedge \mathrm{Entier}(n_{p+1}) \rightarrow$$
$$e(t[\alpha_1, \ldots, \alpha_{i-1}, n_i, n_{i+1} \ldots, n_p], n_{p+1})\ ]\ ]$$

U étant successivement U_0, U_+, U_-.

α_ℓ étant successivement $\sigma_{\ell,0}$, $\sigma_{\ell,+}$, $\sigma_{\ell,-}$ pour $1 \leq \ell \leq i-1$

Il devrait alors être clair pour le lecteur qu'un jeu de tests sur le $H_0 \sqcup H_1$-contexte

$$\langle L_1, S_1, (\mathcal{S}_1), E \rangle$$

acceptable est de la forme :

$$\langle H_0 \sqcup H_1, (\overline{T_i})_{i \in \mathbb{N}} \rangle$$

avec

$$T_i = \{ \; e\left(t[\alpha_1 \cdots \alpha_p], \alpha_{p+1}\right), \quad \alpha_\ell \text{ étant successivement } \sigma_{\ell,0}, \sigma_{\ell,+}, \sigma_{\ell,-} \quad \text{pour } 1 \leq \ell \leq p+1 \; \}$$

et ce, quel que soit i dans $\mathbb{N}^*$.

7.2.7 Discussion

Nous choisissons, pour la brièveté de l'exposé, de ne pas nous étendre sur la description du contexte produit. Elle serait relativement proche de celle présentée en 6.2.7.

La théorie d'hypothèse H_1 produite par notre méthode est extrêmement complexe. De manière essentielle, elle est composée d'hypothèses d'uniformité,

(27), (28) (cf. 7.2.3.)

(42) (cf. 7.2.6)

et d'une hypothèse de régularité, obtenue par stratification de X_{FIle} selon la hauteur des termes considérés

(34) (cf. 7.2.4.)

Rappelons que, pour obtenir un jeu de tests de bonne qualité, il est nécessaire de ne poser de telles hypothèses qu'avec soin, s'assurant pour chacune de ce qu'elle soit "probablement" vérifiée par la structure représentant réellement l'objet sous test.

C'est, nous semble-t-il, le cas pour notre exemple. Cependant, des hypothèses telles que (42) peuvent ne pas être justifiées dans le cas de types plus complexes.

Considérons par exemple le type Ensemble (Entier). Avec des notations évidentes, nous y trouvons un axiome de la forme (cf. [Bidoit 81])

retirer (n, ajouter (n', e)) = <u>si</u> n = n'
<u>alors</u> e
<u>sinon</u> ajouter (n', retirer (n, e)) (43)

Cet axiome sera traduit par la formule suivante :

$$\forall n\, \forall n' [\, \text{Entier}(n) \wedge \text{Entier}(n') \rightarrow [\, n = n' \rightarrow \ldots \,]\,] \quad (44)$$

Si nous appliquons la méthode proposée en 7.2.6, ayant éliminé $\forall n$, nous serons conduit à poser l'hypothèse suivante, parmi d'autres :

$$\exists n' [\, U_+(n') \wedge [\, v_+ = n' \rightarrow \ldots \,]\,] \rightarrow \forall n' [\, U_+(n') \rightarrow [\, v_+ = n' \rightarrow \ldots \,]\,] \quad (45)$$

Or, il est bien évident que cette hypothèse <u>n'est pas justifiée</u>. En effet, elle est trivialement valide, si n' est différent de v_+ (ce qui a toutes les chances de se produire), et n'apporte donc pas réellement d'information.

En fait, n' devrait être astreint à être égal à v_+, plus généralement, à forcer la validité du premier membre de l'implication.

Une hypothèse d'uniformité raisonnable serait donc : 7.40

$$\exists n' \; [\, U_+(n') \wedge n' = v_+ \wedge [\, v_+ = n' \rightarrow \ldots \,]\,] \rightarrow \forall n' \; [\, U_+(n') \rightarrow [\, v_+ = n' \rightarrow \ldots \,]\,] \qquad (46)$$

Plus généralement, ce phénomène se produit pour les présentations ayant l'une des propriétés suivantes.

- L'un des axiomes du type d'intérêt contient un si_alors_sinon donc la condition fait intervenir deux variables distinctes, au moins, de types inférieurs.
- L'une des opérations du type d'intérêt est soumise à une précondition faisant intervenir au moins deux variables de types inférieurs.

C'est le cas de Ensemble (Entier), mais non de File (Entier) ni de Pile (Entier).

Pour pouvoir tester les axiomes de types complexes tels que Ensemble, il faudra, en général, <u>transformer la présentation</u>. Certaines méthodes ont été proposées, notamment par PRIDOIT [Prido 81], pouvant sans doute être appliquées à notre problème (cf. 8.5. et 9.3.)

7.2.8. Description du jeu de tests produit.

Le jeu de tests produit par notre méthode est tout à fait comparable à celui obtenu en 6.2.7.4.

Il est de la forme $\mathcal{T}_i < H_0 \sqcup H_1, (T_i)_{i \in \mathbb{N}} >$

Fixons $i \geq 0$, et décrivons T_i.

Ce test est composé d'une première série d'expériences, décrites en 7.2.3 :

enlever (ajouter (0, file_vide ()) = file_vide ()
enlever (ajouter (u_+, file_vide ()) = file_vide ()
enlever (ajouter (u_-, file_vide ()) = file_vide ().

Il contient une seconde série d'expériences, de la forme :

$$e\,(\,t\,[\alpha_1 \ldots \alpha_{p_t}], \alpha_{p_t+1}\,)\ ,$$

où $t[x_1 \ldots x_{p_t}]$ est un terme bien formé, au sens des types abstraits, de File (Entier), contenant les symboles : enlever, ajouter, file_vide, et les variables : $x_1 \ldots x_{p_t}$ de type Entier, de hauteur inférieure ou égale à i, α_ℓ valant successivement $v_{\ell,0}$, $v_{\ell,+}$, $v_{\ell,-}$, pour $1 \leq \ell \leq p_{t+1}$.

Les u et les v sont des individus de X_{Entier} <u>fixés pour le jeu de tests</u>, tels que :

$U_0(u_0)$, $U_+(u_+)$, $U_-(u_-)$

$U_0(v_{\ell,0})$, $U_+(v_{\ell,+})$, $U_-(v_{\ell,-})$

e(f,n) étant la formule

$$\neg\, f = \text{file\_vide}() \rightarrow \text{enlever}(\text{ajouter}(n, f)) = \text{ajouter}(n, \text{enlever}(f)) \qquad (40)$$

7.2.9. Conclusion

Dans cette partie, nous avons construit, à partir du H_0-contexte initial défini dans la partie précédente $\mathcal{C}_0 = \langle L_0, S, (\mathcal{S}_0), A \rangle$, un $H_0 \sqcup H_1$-contexte $\mathcal{C}_1 = \langle L_1, S, (\mathcal{S}_1), A \rangle$, restriction conservative de $\mathcal{C}_0$, admettant un jeu de tests (finiment) acceptable $\mathcal{C} = \langle H_0 \sqcup H_1, (T_i)_{i \in \mathbb{N}} \rangle$

Cette construction est strictement analogue, quoique beaucoup plus complexe, à celle proposée en 6.2.

Le fait qu'elle soit réalisée dans le cadre du test d'un axiome sur une algèbre n'intervient pas directement dans la méthode, mais intervient de manière cruciale dans la justification des hypothèses de construction H_1.

Les hypothèses que nous avons posées ne sont en fait justifiées que par la forme très particulière des axiomes du type Fils (Entier). Dans le cas de types plus complexes, Ensemble (Entier) par exemple, notre méthode ne serait pas applicable, et des hypothèses plus fines devraient être proposées.

Le problème n'est pas tant lié aux axiomes eux-mêmes qu'aux "structures de contrôle" mal maîtrisées qu'ils utili-

sent (si_alors_sinon, préconditions). Des méthodes 7.44
de transformation formelle de spécification, telles que celles proposées en [Bidoit 81] devraient donc permettre de résoudre, ou plutôt de contourner ce problème.

7.3. Phase d'application et optimisation

De manière analogue à la partie 6.3, après avoir construit un jeu de tests acceptable, nous en étudions les améliorations possibles. Ces améliorations peuvent être distinguées par les axiomes d'hypothèses qu'elles mettent en jeu. Nous considérons donc les optimisations de degré 0, dépendant de H_0, considérées comme "légitimes"; celles de degré 1, dépendant des hypothèses de construction H_1, considérées comme "probablement légitimes"; celles de degré 2, faisant intervenir de nouvelles hypothèses H_2, considérées comme "suspectes".

Dans cette partie, comme dans les précédentes, nous n'insisterons pas sur les points non spécifiques du problème du test d'un axiome sur une algèbre.

7.3.1. Etude du jeu de tests produit.

Examinons tout d'abord le jeu de tests produit lors de la phase précédente, afin de mieux percevoir les améliorations possibles et nécessaires.

De manière grossière, le test de niveau i est constitué d'expériences résultant de l'instanciation des variables de l'axiome sous test par des termes fermés.

Les variables de type inférieur sont instanciées par des <u>individus représentatifs</u> des algèbres supportant ces types.

Les variables du type d'intérêt sont instanciées par des <u>termes fermés</u> construits à partir d'une <u>famille "génératrice"</u> d'opérations de ce type et d'individus de types inférieurs représentatifs.

Dans notre exemple, par l'hypothèse de complétude suffisante (cf. 7.1.3.), la famille {enlever, ajouter, file_vide} est génératrice pour le type File (entier).

Sans cette hypothèse, conservant seulement celle de primarité de X en tant que File (Entier)-algèbre, l'opération "premier" aurait aussi dû intervenir dans la construction.

En réalité, le test T_i est composé d'instanciations des matrices des axiomes non-logiques (uniques) des théories B et C (cf. 7.2.1), soit : 7.47

File(f) →
  Entier(n) → (47)
    f = file_vide() →
      enlever(ajouter(n, f)) = file_vide()

File(f) →
  Entier(n) → (48)
  ¬ f = file_vide() →
      enlever(ajouter(n, f)) = ajouter(n, enlever(f)).

Ecrivons un exemple de ces instanciations dans le cas de T_3 (les individus de type Entier "représentatifs" de leur domaine U_0, U_+, U_- sont arbitraires, cf. 6.2.7.4).

① <u>instanciations de (47)</u>

f = file_vide() n = 0 (49)

n = -5

n = 12

② <u>instanciations de (48), hauteur = 0</u>

il n'existe pas de terme de hauteur 0 (50)

③ <u>instanciations de (48), hauteur = 1</u>

$f = \text{file\_vide}()$ $\quad n = 0$

$n = -43$ $\qquad$ (51)

$n = 5$

④ <u>instanciations de (48), hauteur = 2</u>

$f = \text{ajouter}(n_1, \text{file\_vide}())$

$n_1 = 0 \quad n = 0$

$n = -10$

$n = 7$

$n_1 = -2 \quad n = 0$ $\qquad$ (52)

$n = -13$

$n = 15$

$n_1 = 3 \quad n = 0$

$n = -8$

$n = 3$

$f = \text{enlever}(\text{file\_vide}())$ $\quad n = 0$

$n = -93$ $\qquad$ (53)

$n = 12$

⑤ <u>instanciations de (48), hauteur = 3</u>

$f = \text{ajouter}(n_1, \text{ajouter}(n_2, \text{file\_vide}()))$ $\qquad$ (54)

$n_1 = 0 \quad n_2 = 0 \quad n = 0$

$\vdots$

$f = \text{ajouter}(n_1, \text{enlever}(\text{file\_vide}()))$ $\qquad$ (55)

$n_1 = 0 \quad n = 0$

$\vdots$

f= enlever (ajouter (n_1, file_vide ())) (56)

$n_1 = 0$ $n = 0$

⋮

f= enlever (enlever (file_vide ())) (57)

$n = 0$

$n = -15$

$n = 33$

La taille de T_i croît exponentiellement avec i, mais de nombreuses optimisations sont possibles, comme le montrent les paragraphes suivants.

7.3.2. Optimisations de degré 0

Comme nous l'avons vu en 6.3.1 et en 6.3.2, les optimisations de degré 0 (virtuelles éventuellement) permettent essentiellement :

- de manipuler trivialement le jeu de tests (manipulation d'indice, élimination des expériences triviales etc...),
- d'augmenter la redondance du jeu de tests.

Une optimisation intéressante ici est l'élimination de l'instanciation (51), puisque l'on peut démontrer :

$$\neg \text{file\_vide}() = \text{file\_vide} \rightarrow \ldots$$

quel que soit le second membre.

Dans des cas plus complexes que celui de notre exemple, nous serons amené à augmenter la redondance du jeu de tests.

Par exemple, si l'entier 5 (en fait, succ(succ(succ(succ(succ 0))))) joue un rôle crucial, nous pourrons instancier systématiquement les ni et n par 5, comme nous le faisons déjà pour 0.

Ce serait en particulier le cas si 5 était une opération de la présentation du type Entier, au même titre que 0. Il ne serait

pas raisonnable de traiter différemment ces constantes.

De même, si certains individus de XFile nous paraissent d'un intérêt primordial, nous instancierons systématiquement f par cette valeur, en plus des instanciations précédentes.

Conformément à 6.3.2., toutes ces optimisations doivent être considérées comme tout à fait "légitimes".

7.3.3. Optimisations de degré 1

Selon 6.3.3., ces optimisations permettent de réduire la redondance d'un jeu de tests, ou d'en modifier les expériences.

Une optimisation pourrait par exemple être, dans notre cas, de n'instancier les variables de type Entier que par 0, 1 et -1. En théorie, ce jeu de tests est tout aussi acceptable que celui obtenu par des instanciations plus ou moins "aléatoires".

Peu de praticiens se contenteraient cependant d'un tel jeu de tests, pourtant formellement absolument équivalent à celui esquissé en 7.3.1.

C'est que, en fait, les hypothèses d'uniformité (42) utilisées en 7.2.6, et celles (27) (28) décrites en 7.2.3. ne sont que "probablement" vérifiées par l'objet sous test, et il est imprudent pour un jeu de tests de trop en dépendre. Le fait d'"échantillonner" les valeurs représentatives des sous-domaines d'uniformité donne au jeu de tests une plus grande stabilité, une plus grande indépendance vis à vis des hypothèses d'uniformité.

Une optimisation souhaitable sera donc

de choisir des valeurs représentatives différentes à chaque instanciation, de manière à "échantillonner" le plus de valeurs possibles dans le sous-domaine considéré.

7.3.4. Optimisations de degré 2

Conformément à 6.3.4., ces optimisations font intervenir des connaissances, des hypothèses supplémentaires concernant l'objet sous test.

Notre exemple consiste à tester l'axiome ⊕ du type File sur la File(Entier)-algèbre X. En général, ce genre de test aura pour but de montrer que X est une E_{File}-algèbre, où E_{File} désigne l'ensemble des axiomes du type File (cf. l'introduction à 7.1.). Nous aurons donc, par exemple, déjà testé que X valide l'axiome ⊗ :

enlever (file_vide ()) = file_vide ()

(un jeu de tests serait de toute façon rapidement construit, puisque cette formule est effectivement vérifiable sur les contextes considérés).

Supposons donc que, de manière préméditée ou non, nous sachions que la structure représentant réellement l'objet sous test valide la formule :

enlever (file_vide ()) = file_vide (). (58)

Formons alors H_2, théorie des hypothèses d'optimisations :

$$H_2 := \{ (58) \}.$$

Effectuons une restriction (non-conservative!) pour obtenir un $H_0 \cup H_1 \sqcup H_2$ - contexte $\mathcal{C}_2$. Par apport à ce contexte, notre jeu de tests peut être optimisé par la suppression de toutes les instanciations de f de la forme :

enlever(... (enlever (file_vide ()) ...)

ou de toutes celles contenant un sous-terme de cette forme : (53), (55), (57).

En effet, l'expérience correspondante est démontrée par celle faite avec file_vide () (ou en remplaçant le sous-terme concerné par file_vide ()), les hypothèses d'uniformités et (58).

Plus généralement, si nous connaissons <u>a priori</u> une famille "génératrice" pour les individus de X_{File}, nous pourons nous contenter d'instancier les variables de type File par des termes construits sur cette famille.

Supposons, par exemple, que nous sachions que { ajouter, file_vide } engendre X_{File} sur X_{Entier}.

Pour chaque terme fermé contenant le symbole "enlever", nous aurons un axiome nous fournissant sa forme réduite. Par exemple :

enlever(ajouter (3, file_vide ())) = ajouter (2, file_vide ())

Soit H_2 l'ensemble (infini) de ces axiomes. Comme précédemment, opérons une restriction (non-conservative) pour obtenir un $H_0 \cup H_1 \cup H_2$ - contexte $\mathcal{C}_2$. Par rapport à ce contexte, notre jeu de tests est formellement asymptotiquement équivalent à celui obtenu en supprimant toutes les instanciations de variables de type File contenant le symbole "enlever" :

(53), (55), (56), (57).

Ces optimisations sont donc extrêmement efficaces, mais font intervenir des connaissances sur la structure de X extérieures à notre problème.

7.3.5. Conclusion

Cette partie nous a permis d'examiner diverses optimisations du jeu de tests construit précédemment.

Les techniques employées, comme en 7.2., ne sont pas fondamentalement différentes de celles utilisées dans l'exemple du chapitre précédent (cf. 6.3.).

Cependant, le fait que le problème initial soit exprimé dans le formalisme des types abstraits algébriques nous permet de décrire, de manière naturelle et succinte, des optimisations efficaces et légitimes.

Il nous permet d'autre part d'exprimer simplement des conditions sur l'objet concerné conduisant à des optimisations du degré 2 très intéressantes. De telles conditions sont souvent vérifiées dans la pratique dans la mesure où l'on ne s'intéresse généralement qu'à de "bonnes" (ou supposées telles) algèbres.

7.4. Conclusion

Le chapitre précédent avait pour objet de montrer l'application du patient travail théorique qui le précède sur un exemple concret, le test d'un programme de tri.

La méthode que nous avions alors employée, notamment lors de la phase de représentation, avait pu paraître lourde et "patauude", inélégante quoique rigoureuse.

Ce chapitre démontre qu'elle est, en fait, particulièrement bien adaptée à un problème énoncé dans le formalisme des types abstraits algébriques, typiquement le test d'un axiome sur une algèbre.

La phase de représentation n'est qu'un jeu d'écriture, de traduction entre les types abstraits algébriques et la logique égalitaire du premier ordre. Par quelques heuristiques simples, préconditions et si_alors_sinon (et cas d'échec éventuellement) peuvent être intuitivement correctement pris en compte.

La phase de construction utilise les méthodes générales, de manière souvent plus efficace et naturelle. Nous pouvons en effet confondre termes de la logique et termes

d'un type abstrait. D'autre part, le formalisme des types abstraits algébriques nous permet d'énoncer des conditions précises concernant la validité de certaines hypothèses d'uniformité.

La phase d'application utilise elle aussi les méthodes générales d'optimisation. Nous pouvons, de par la forme très particulière du problème posé et des motivations qui le sous-tendent, énoncer des conditions naturelles, souvent vérifiées en pratique, conduisant à des optimisations de degré 2 intéressantes et rentables

Ces avantages pratiques s'ajoutent donc aux avantages théoriques décrits dans l'introduction à ce chapitre (dénombrabilité, calculabilité, universalité etc.) pour confirmer le domaine des types abstraits algébriques comme terrain d'application privilégié de notre travail.

Chapitre 8 : Validation de spécifications abstraites par test.

Le chapitre précédent a montré que le formalisme des types abstraits algébriques constitue un domaine d'application extrêmement privilégié pour notre travail.

Considérant une File (Entier)-algèbre X, et un axiome du type File, nous avons décrit le processus de test de cet axiome sur l'algèbre X. La famille de tests exhibée est en fait constituée de tests aux structures relativement simples. Le test d'indice i, Ti, est formé de toutes les formules fermées dérivables de l'axiome considéré par l'instanciation de ses variables à l'aide de termes de hauteur au plus égale à i (cf. 7.2.8. et 7.3.1.).

Ce jeu de tests est construit sur des hypothèses relativement fortes mais cependant très intuitives. Nous sommes en particulier capable de décrire par des conditions suffisantes, de manière précise leur domaine de validité (cf. 7.2.7.)

Ces remarques nous permettent d'envisager une généralisation de cet exemple au problème très général de la <u>validation de spécifications abstraites</u>, exprimées dans le formalisme des types abstraits algébriques, par test. La nature essentiellement récursive du jeu de tests produit nous permet d'envisager <u>l'implantation d'un outil automatique</u> réalisant une telle fonction.

Ce chapitre est consacré à la description d'un prototype d'un tel outil, fonctionnant actuellement sur le système MULTICS de l'INRIA, à Rocquencourt (FRANCE).

Cet outil est actuellement capable de traiter des spécifications de complexité analogue à celle présentée au chapitre précédent.

Il peut être rapproché des outils proposés par J. GANNON & al. (DAISTS, cf. [GMHA 80] et [GHM 81]), et de ceux construits par B. HOUSSAIS notamment dans le cadre de la validation des compilateurs par test (cf. [Boug 82])

8.1. Principes fondamentaux

Nous décrivons dans cette partie les principes fondamentaux sous-jacents à la réalisation et au fonctionnement de notre outil.

Des justifications théoriques précises pourraient être données, concernant notamment l'existence d'un jeu de tests sous les hypothèses que nous énonçons ci-dessous. La démonstration, qui généralise et éclaire la méthode employée en 7.2., serait cependant trop longue pour être reproduite ici. Que le lecteur se convainque seulement qu'une telle démonstration est possible.

De même, dans tout ce chapitre, nous nous permettrons quelques abus de langage, confondant en particulier des notions issues de la logique du premier ordre et d'autres issues du formalisme des types abstraits algébriques, lorsque nous considérerons que cela rend la compréhension des choses plus intuitive. Que le lecteur sache cependant qu'il pourrait, à chaque fois, trouver une expression rigoureuse correspondante.

8.1.1. Position du problème

Généralisant l'exemple proposé au chapitre précédent (cf. 7.1.1.), le problème est ici le suivant.

En reprenant les notations de [Bidoit 81], considérons une spécification abstraite, c'est-à-dire essentiellement la donnée d'une signature Σ et d'un ensemble d'axiomes, E. Considérons une Σ-algèbre X. Le problème est de décider si X est une Σ,E-algèbre ou non.

Autrement dit, il s'agit de valider les axiomes de E sur l'implantation X.

Pour des raisons techniques, nous ne considérons que le cas où Σ est une signature finie, et E un ensemble au plus dénombrable d'axiomes.

Ces axiomes peuvent contenir des "structures de contrôle" classiques, en particulier des "si_alors_sinon". Notre outil est bien adapté à des axiomes sous forme de "clause de HORN universelle"

$$t_1 = t'_1 \wedge \ldots \wedge t_n = t'_n \rightarrow t = t'$$

ou $t_1 = t'_1 \wedge \ldots \wedge t_n = t'_n \rightarrow \neg t = t'$.

D'autre part, nous admettons que la spécification astreigne certaines opérations à des préconditions ou cas d'échec (cf. [Gutt 80], [Gaud 80], [Prido 81]).

Par des arguments dérivés du théorème de localisation (cf. 5.2.), pour valider l'ensemble d'axiomes E sur la Σ-algèbre X, il suffit de savoir le faire successivement pour chacun des axiomes de E.

Nous sommes donc ramené au problème présenté au chapitre 7, qui est le test d'un axiome sur une Σ-algèbre.

8.1.2. Hierarchie et certitude

Conformément à [Bidoit 81], nous considérons une spécification hiérarchisée, c'est-à-dire telle que l'on y ait distingué un type d'intérêt.

Dans la pratique, un telle distinction correspond au fait que l'algèbre X a été construite de manière modulaire, par enrichissement successif d'un noyau central (en général, le type Booléen).

La spécification peut donc être divisée en deux parties. L'une concerne les types d'ordre inférieur, et l'autre le type d'intérêt et ses rapports avec les types inférieurs.

De manière générale, l'on peut considérer que la partie inférieure de la spécification, l'environnement, a déjà été validée sur X, lors des étapes précédentes de construction.
Tout se passe donc comme si l'<u>environnement était parfait</u>, comme si X était certainement une "bonne" algèbre vis à vis de cette sous-spécification.

Plus formellement, nous supposons donc que X est <u>une algèbre première</u> vis à vis de cette

partie de la spécification, et que tous les axiomes de celle-ci sont valides sur X.

Rappelons que ceci n'est qu'une hypothèse de modélisation. L'important n'est pas qu'elle soit vérifiée, mais bien que tout se passe comme si elle l'était.

La partie de la spécification correspondant au type d'intérêt (File, au chapitre 7) est considérée par contre comme "suspecte". Cependant, en pratique, l'on ne s'intéresse jamais qu'à de "bonnes" algèbres, sinon à des algèbres "raisonnables" et "utiles".

Formellement, nous interpréterons cette remarque en supposant, lors de la phase de représentation, que X est une algèbre première par rapport à la spécification globale. Rapportée à l'hypothèse précédente, celle-ci signifie que l'extension de X correspondant à l'ajout du type d'intérêt est suffisamment complète.

Là encore, l'important est que tout se passe comme si cette hypothèse était vérifiée. De toute façon, pratiquement, seule la composante première de l'algèbre est accessible à l'utilisateur.

8.1.3. Hauteur d'un terme

En confondant l'algèbre X et ses supports, nous pouvons écrire

$$X = X_{T_{int}} \sqcup X_0 ,$$

où $X_{T_{int}}$ est l'ensemble support du type d'intérêt, et X_0 la réunion de ceux des types inférieurs.

D'après les hypothèses exprimées en 8.1.2, tout individu x du type d'intérêt ($x \in X_I$) peut s'écrire sous la forme d'un terme

$$t\,[y_1 \cdots y_n]$$

où $y_1, \ldots, y_n$ sont des individus des types inférieurs ($y \in X_0$), et $t[\cdot, \ldots, \cdot]$ un terme ne contenant que des opérations du type d'intérêt

Remarquons que, x étant donné, les y et t ne sont en général pas uniques.

La hauteur de $t\,[y_1 \cdots y_n]$ est par définition la hauteur de l'arbre constitué par le terme $t[\cdot, \ldots, \cdot]$.

Il n'y a pas de terme de hauteur 0. Les constantes du type d'intérêt sont des termes de hauteur 1.

Il n'y a qu'un nombre fini d'arbres

$$t\,[\cdot, \ldots, \cdot]$$

de hauteur au plus égale à i.

8.1.4. Régularité et Uniformité

Nous pouvons maintenant décrire les hypothèses utilisées implicitement par notre outil.

Nous avons admis que X est une "bonne" algèbre vis à vis des types inférieurs. Nous pouvons donc attendre des supports d'ordre inférieur une certaine uniformité par rapport aux formules que nous serons amené à utiliser.

Plus précisément, soit Type un type d'ordre inférieur, et X_{Type} son support. Nous supposons que nous connaissons <u>a priori</u> un recouvrement de X_{Type} en sous-domaines d'uniformité $U_1, \dots, U_n$. La formule suivante est donc valide :

$$\forall y \; [\; Type(y) \leftrightarrow U_1(y) \vee \dots \vee U_n(y) \;]$$

Considérons maintenant une formule de la forme

$$\forall y \; [\; U_i(y) \rightarrow \phi(y) \;] \qquad 1 \leq i \leq n$$

y étant la seule variable libre de ϕ, de type Type.

Nous posons l'hypothèse d'uniformité suivante

$$\exists y \, [U_i(y) \wedge \phi(y)] \rightarrow \forall y \, [U_i(y) \rightarrow \phi(y)]$$

et ce pour <u>toutes les formules ϕ que nous rencontrerons de cette forme</u>

Pour tester la formule

$$\forall y \, [\, \mathrm{Type}(y) \rightarrow \phi(y) \,]$$

il suffira donc de choisir $y_1, \ldots, y_n$ éléments de X_{Type}, appartenant respectivement à $U_1, \ldots, U_n$, et de réaliser les expériences

$$\phi(y_1), \ldots, \phi(y_n)$$

Par contre, nous avons considéré que X est une algèbre "suspecte" vis à vis du type d'intérêt, que nous notons Tint. Il n'est donc plus "raisonnable" de supposer que X_{Tint} est uniforme vis à vis des formules que nous rencontrerons. Nous devons donc poser non pas des hypothèses d'uniformité, mais de régularité.

Comme au chapitre 7, nous envisageons des hypothèses de régularité sur la hauteur des termes considérés. Dans la pratique, la hauteur des termes impliqués étant le plus souvent petite, il suffit de choisir une constante k virtuelle assez grande pour que l'hypothèse soit "raisonnable". L'hypothèse de primarité et la remarque du 8.1.3. montrent que l'hypothèse est valide si "$k \simeq \infty$".

Considérons donc un formule

$$\forall x \, [\, \mathrm{Tint}(x) \rightarrow \phi(x) \,],$$

où x est la seule variable libre de ϕ, de type

Tint. Nous introduisons l'hypothèse suivante

$$\bigwedge_{\text{hauteur}(t) \leq k} d_t \longrightarrow \forall x \, [\mathrm{Tint}(x) \rightarrow \phi(x)]$$

avec $d_t = \forall y_1 \cdots \forall y_p$

$$[\mathrm{Type}_1(y_1) \wedge \cdots \wedge \mathrm{Type}_p(y_p) \longrightarrow \phi(t[y_1 \cdots y_p])]$$

Rappelons qu'il n'y a qu'un nombre fini de termes t de hauteur au plus égale à k.

Pour chaque formule ϕ de ce type que nous utiliserons, nous poserons une telle hypothèse.

Le test de

$$\forall x \, [\mathrm{Tint}(x) \rightarrow \phi(x)]$$

se ramène donc à celui de chacun des d_t (cf. 7.3.4.).

8.1.5. Conclusion

Nous avons décrit dans cette partie les principes généraux sur lesquels est bâti notre outil.

Ces principes généralisent en fait, de manière triviale, ceux énoncés dans le cas particulier du type File, au chapitre 7.

Cependant, comme le précise 7.2.7., ces principes, et notamment les hypothèses d'uniformité énoncées en 8.1.4., ne sont valides que pour une certaine classe de spécifications.

Les axiomes du type d'intérêt doivent être tels que les variables des types inférieurs soient uniformes, de manière indépendante les unes des autres, vis à vis d'eux.

En particulier, ils ne doivent pas contenir de si_alors_sinon dont la condition fasse intervenir deux variables distinctes de types inférieurs.

De même, les préconditions et cas d'échec ne devraient pas dépendre de deux variables distinctes de types inférieurs.

Le type Ensemble, par exemple, n'est donc pas directement envisageable, à cause de l'axiome

retirer (n, ajouter (n', e)) =
  <u>si</u> n = n' <u>alors</u> e
    <u>sinon</u> ajouter (n', retirer (n, e))

Nous citerons plus loin quelques méthodes permettant de se ramener à des spécifications moins complexes (cf. 8.5.).

8.2. Description du jeu de tests

Dans cette partie, nous présentons le test produit par notre outil dans le cas de la validation d'un axiome sur une algèbre X.

Soit $e(x_1, \ldots, x_m, y_1, \ldots, y_n)$ cet axiome, les x étant des variables du type d'intérêt, les y étant des variables de types inférieurs

Selon la forme des hypothèses présentées en 8.1.4., une famille de tests $(T_i)_{i \in \mathbb{N}^-}$ associée à un jeu de tests acceptable sur le contexte obtenu à partir d'un contexte initial par restriction conservative guidée par ces hypothèses (cf. 5.1.3.), est donnée par la définition suivante (cf. 7.2.8.).

L'indice i étant donné, T_i est l'ensemble des expériences de la forme suivante :

$$e(t_1[y_1^1 \ldots y_{p_1}^1], \ldots, t_m[y_1^m, \ldots, y_{p_m}^m], y_1, \ldots, y_n)$$

où $t_1, \ldots, t_m$ sont des termes de hauteur au plus égale à i, et les y des individus de type convenable, appartenant successivement à chacun des sous-domaines d'uniformité U définis en 8.1.4.

8.2.1. Opérations pures et méta-opérations

Tout au long de ce chapitre et du précédent, nous avons soigneusement distingué les termes de la signature de la spécification, et les individus de l'algèbre sur laquelle est testée cette spécification.

Par primarité, tout individu est la valeur d'un terme. Dans le cas des constantes, on identifie l'opération et l'individu qui en est la valeur.

Au chapitre 7, en 7.2.3. et 7.2.6., nous avons ajouté de nouveaux symboles au langage, les symboles u et s, dont la valeur, dans chacune des structures de la famille (S_i) est un individu "représentatif" du sous-domaine d'uniformité auquel il se rapporte.

Nous appelons ces symboles des <u>méta-opérations</u>, en ce sens qu'ils servent à désigner un individu d'une algèbre donnée, et non à construire des termes.

Les autres symboles sont dits <u>opérations pures</u>.

A la différence des opérations pures, l'ajout de méta-opérations <u>ne change pas</u> la spécification, car elles sont liées à l'algèbre considérée,

plus précisément aux individus de l'algèbre, et non aux opérations qui opèrent sur ceux-ci.

En logique du premier ordre, si $\mathcal{S}$ est une structure d'un langage L sur un ensemble S, l'on peut ajouter à $\mathcal{S}$ des méta-opérations ayant pour noms les noms des individus de S. $\mathcal{S}$ est alors étendue canoniquement en une L(S)-structure. Les symboles de L sont les opérations pures, celles de S les meta-opérations (cf. annexe 3).

Ces méta-opérations ont déjà été considérées en [Gaud 80]. Elles sont alors notées ainsi.

Ent "41"

désigne l'individu 41 du support du type Ent dans l'algèbre considérée, $\mathbb{Z}$.

En particulier, Ent "0" et zero () jouent des rôles très différents. Pour notre outil, le premier sera implanté par

(quote 0)

et le second par

(zero), avec

(defun zero () 0)

Si $U_1 \ldots U_n$ sont les sous-domaines associés à l'ensemble X_{Type}, où Type est un type d'ordre inférieur, nous notons

Type_1, ..., Type_n

les méta-opérations associées.

8.2.2 Classification des opérations pures

Parmi les opérations pures du type Tint d'intérêt, nous distinguons plusieurs classes:

- Les opérations <u>constantes</u>, de profil
  $\longrightarrow$ Tint
- les opérations <u>croissantes</u>, de profil
  $Type_1 \times \ldots \times Type_n \rightarrow Tint$
  où $Type_\ell$ est un type d'ordre inférieur, $1 \leq \ell \leq n$
- les opérations <u>décroissantes</u>, de profil
  $T_1 \times \ldots \times T_n \rightarrow Type$
  au moins l'un des type du domaine étant le type d'intérêt, Type étant d'ordre inférieur
- les opérations <u>étales</u>, de profil
  $T_1 \times \ldots \times T_n \rightarrow Tint$
  au moins l'un des type du domaine étant le type d'intérêt.

En fait, plus généralement, nous nous donnerons un ordre total sur les types de la spécification, et les opérations de chacun des types seront ainsi réparties. Ceci peut amener éventuellement à déplacer une opération d'un type dans un autre (supérieur, en général) afin de permettre de classer ainsi toutes les opérations proposées.

8.2.3. Forme générale des termes

Soit i un entier donné.

Examinons, par 8.1.3., la forme des termes de hauteur au plus égale à i.

Si i est inférieur strictement à 1, il n'y en a pas.

Si i est 1, ceux-ci sont de l'une des deux formes suivantes :

- $f[\,]$, où f est une opération <u>constante</u> de Tint
- $f[\cdot, \ldots, \cdot]$, où f est un opération <u>croissante</u> de Tint

Si i est supérieur strictement à 1, ceux-ci peuvent être des deux formes précédentes, ou de la forme $f\,[\,t_1[\cdot, \ldots, \cdot], \ldots, t_m[\cdot, \ldots, \cdot], \cdot, \ldots, \cdot\,]$, où f est une opération <u>étale</u> de Tint, de profil $\mathrm{Tint} \times \ldots \times \mathrm{Tint} \times \mathrm{Type}_1 \times \ldots \times \mathrm{Type}_n \to \mathrm{Tint}$, les Type_e étant des types d'ordres inférieurs,

$$t_1[\cdot, \ldots, \cdot], \ \ldots, \ t_m[\cdot, \ldots, \cdot]$$

étant des termes de hauteurs au plus égales à $i-1$.

Par primarité, tout individu x de X_{Tint} est la valeur d'un terme, de hauteur convenable, de la forme précédente.

Les opérations <u>décroissantes</u> ne jouent donc 8.19
aucun rôle à ce niveau.

8.2.4 Optimisation

Nous avons ainsi décrit une famille de tests $(T_i)_{i \in \mathbb{N}}$ formant, avec les hypothèses exposées en 8.1.4, un jeu de tests acceptable.

Notre outil n'utilise en fait pas ce jeu de tests, mais une optimisation de celui-ci, analogue à celle proposée en 7.3.2.

Cette optimisation est de degré 0, et n'altère donc pas la qualité du jeu de tests considéré.

Même si, au niveau conceptuel comme au niveau technique, opérations constantes et meta-opérations sont de natures profondément différentes (cf. 8.2.1.), ces objets se manipulent de la même façon au niveau de l'instanciation combinatoire des axiomes.

D'autre part, la spécification ayant été écrite de main humaine selon une intuition précise, les constantes désignent probablement des <u>valeurs sensibles</u>, <u>critiques</u>, qu'il serait regrettable de négliger.

Au contraire, les méta-opérations s'évaluent probablement en des <u>individus représentatifs</u>, <u>génériques</u> de l'algèbre considérée.

Au niveau pratique, un "bon" jeu de tests doit tenir compte à égalité de ces deux aspects, combinant ainsi les heuristiques du test par échantillonage aléatoire, et du test aux valeurs spéciales (cf. [Boug 82]).

Dans le processus d'instanciation de l'axiome considéré, nous utiliserons donc concurremment opérations constantes et méta-opérations à chaque fois que l'un ou l'autre type d'instanciation sera proposé par l'algorithme (cf. la partie centrale de l'algorithme d'instanciation, proposé en annexe 4).

8.2.5. Conclusion

Nous avons présenté dans cette partie le jeu de tests utilisé par notre outil pour la validation d'un axiome sur une algèbre.

Fondamentalement, les tests de celui-ci sont constitués de toutes les instanciations fermées de l'axiome considéré, de hauteur au plus égale à l'indice du test.

La théorie que nous avons longuement développée au long de ce travail nous a permis de donner des justifications précises des choix qui ont conduit à cette forme. Elle nous permet, dans une certaine mesure, d'évaluer la qualité de l'optimisation empirique réalisée en confondant opérations constantes et méta-opérations.

Cette capacité à guider l'intuition, à la vivifier, la féconder, à la contrôler, la justifier, à lui proposer un langage et des concepts pour s'exprimer, nous paraît être le meilleur critère de jugement de ce travail.

8.3. Notion de candidat

Avant de décrire de manière précise notre outil, il nous faut dire quelques mots du problème des structures de contrôle.

Par structure de contrôle, nous entendons les objets intervenant dans une spécification permettant d'établir des liaisons entre les termes, plus complexes que celles obtenues par le symbole d'égalité.

Dans la plupart des cas, ce sont

- les "si_alors_sinon"
- les préconditions
- les cas d'échec

(cf. [Gutt 80], [Gaud 80], [Bido 81])

Nous incluons aussi dans ces classes les symboles "$\wedge$" et "$\rightarrow$" qui permettent de construire des spécifications dites conditionnelles (positives).

Cependant, nous ne traitons pas explicitement ces deux derniers cas, puisque de telles constructions reviennent à astreindre les axiomes considérés à des contraintes. Plus précisément, la forme

$$t_1 = t'_1 \wedge \ldots \wedge t_n = t'_n \longrightarrow t = t'$$

peut se ramener à cette forme <u>d'exigence</u> (voir plus loin)

$$\begin{cases} \text{contraintes} : & t_1 = t'_1, \dots, t_n = t'_n \\ \text{termes} : & t = t' \end{cases}$$

La partie "termes" ne doit être prise en compte que si la partie "contraintes" est valide sur l'algèbre considérée.

Il en est de même pour la forme appellée "clause de HORN universelle"

$$t_1 = t'_1 \,\&\, \dots \,\&\, t_n = t'_n \Rightarrow \neg\, t = t'$$

8.3.1. "si_alors_sinon" et contraintes

Le problème du "si_alors_sinon" est dû à l'appel par necessité.

Il n'est pas possible d'instancier directement un axiome contenant un "si_alors_sinon" si des préconditions au cas d'échec éventuels peuvent être mis en jeu.

Considérons par exemple le cas du type File, présenté en introduction à la partie 7.1. Examinons l'axiome ③

premier (ajouter (n, f)) = si f = file_vide ()
alors n (1)
sinon premier (f)

avec la précondition

premier (f) : ¬ (f = file_vide ()) (2)

Instancions l'axiome (1) par f = file_vide () et n = 0 par exemple.

premier (ajouter (0, file_vide ())) = si file_vide () = file_vide ()
alors 0
sinon premier (file_vide ())

Mais la précondition (2) n'étant pas satisfaite par la branche "sinon", cet axiome ne devrait pas être considéré.

l'instanciation de (1) par f = file_vide()
(et n = 0) est donc illégitime

En fait, il est manifeste que cette méthode de raisonnement conduit à des résultats incorrect. Pour l'instanciation considérée, (1) se réécrit naturellement en

premier (ajouter (0, file_vide()) = 0 ,

et (2) est satisfaite probablement.

Pour simuler l'appel par nécessité du "si_alors_sinon" lors de l'appel par valeur réalisé par une instanciation, il nous faut diviser (1) en deux sous-axiomes, en nous inspirant de 7.1.4. D'où les sous-axiomes suivant

premier (ajouter (n, f)) = n (3)

premier (ajouter (n, f)) = premier (f) (4)

Pour savoir quelle équation doit être prise en compte, il nous faut leur associer des <u>contraintes</u> d'utilisation, dérivées de (1). D'où les spécifications suivantes

$$\begin{cases} \text{contraintes} : & f = \text{file\_vide}() \\ \text{termes} : & \text{premier}(\text{ajouter}(n, f)) = n \end{cases} \quad (5)$$

$$\begin{cases} \text{contraintes} : & \neg f = \text{file\_vide}() \\ \text{termes} : & \text{premier}(\text{ajouter}(n, f)) = \text{premier}(f) \end{cases} \quad (6)$$

Nous retrouvons donc l'idée extrêmement classique consistant à "pré conditionner" les règles de réécriture d'un type abstrait algébrique, isolant ainsi la partie "utile" de l'axiome de sa partie "contrôle"

Lors de la validation de (4) sur l'algèbre considérée, nous nous ramènerons au problème de la validation de (5) et (6). Etant donné une instanciation de n et de f, nous évaluerons d'abord la partie "contraintes".

Si celle-ci s'évalue en F (nil habituellement en LISP), le sous-axiome n'a pas à être considéré. Il est <u>sans signification</u>.

Si celle-ci s'évalue en T (true en LISP), alors la partie "termes" est évaluée; si elle s'évalue en T, le sous-axiome est valide; sinon, il ne l'est pas.

L'axiome initial est alors valide si tous les sous-axiomes <u>significatifs</u> pour l'instanciation considérée le sont.

La donnée de <u>contraintes</u> et de <u>termes</u> sera appelée un <u>candidat</u>

8.3.2. Préconditions

Examinons maintenant le problème des préconditions, selon l'heuristique proposée en 7.1.4.

La difficulté tient à ce que, dans la spécification abstraite, ce sont les opérations qui sont soumises aux préconditions, et non les axiomes.

Or, étant donné une formule à valider sur une algèbre, il nous faut savoir <u>a priori</u> si les termes y intervenant sont légitimes ou non. S'ils ne le sont pas, l'évaluation de la formule sur l'algèbre n'a pas de sens. Autrement dit, il nous faut déduire, des préconditions portant sur des opérations, des préconditions portant globalement sur l'axiome à valider, et plus précisément sur les <u>candidats</u> déduits de cet axiome selon 8.3.1.

Nous choisissons arbitrairement, en l'absence d'une sémantique rigoureuse de la notion de précondition (cf. 7.1.4.), de ne considérer que celles issues du champ "termes" du candidat considéré, autrement dit de la partie <u>utile</u> de l'axiome.

Les termes intervenant dans les autres

champs peuvent donc éventuellement être illégitimes, auquel cas nous laissons le comportement de notre outil plus ou moins indéfini (cf. 8.4.6.)

Un candidat sera considéré comme légitime vis à vis des préconditions si tous les termes intervenant dans le champ "termes" satisfont leurs préconditions.

Nous sommes donc conduit à enrichir la notion de candidat d'un champ "précon-ditions" supplémentaire.

L'axiome (1) donne donc naissance à deux candidats.

$$\begin{cases} \text{contraintes :} & f = \text{file\_vide}() \\ \text{préconditions :} & \neg\, \text{ajouter}(n, f) = \text{file\_vide}() \\ \text{termes :} & \text{premier}(\text{ajouter}(n, f)) = n \end{cases} \qquad (7)$$

$$\begin{cases} \text{contraintes :} & \neg\, f = \text{file\_vide}() \\ \text{préconditions :} & \neg\, f = \text{file\_vide}(), \\ & \neg\, \text{ajouter}(n, f) = \text{file\_vide}() \\ \text{termes :} & \text{premier}(\text{ajouter}(n, f)) = \text{premier}(f) \end{cases} \qquad (8)$$

Un candidat n'est valide que si toute instanciation (fermée) telle que les formules des champs "contraintes" et "préconditions" s'évaluent à T (true en LISP) est telle que les formules

du champ "termes" s'évaluent à $\top$, sur l'algèbre considérée.

8.3.3. Cas d'échec

Le problème des cas d'échec est exactement le même que celui des préconditions.

Nous sommes donc conduit à ajouter un nouveau champ "echecs" à la définition d'un candidat.

Les candidats dérivés de l'axiome (1) sont donc de la forme

$$\begin{cases} \text{contraintes : } f = \text{file\_vide}() \\ \text{préconditions : } \neg\, \text{ajouter}(u, f) = \text{file\_vide}() \\ \text{échecs :} \\ \text{termes : } \text{premier}(\text{ajouter}(u, f)) = u \end{cases} \qquad (9)$$

$$\begin{cases} \text{contraintes : } \neg\, f = \text{file\_vide}() \\ \text{préconditions : } \neg\, f = \text{file\_vide}() \\ \qquad\qquad \neg\, \text{ajouter}(u, f) = \text{file\_vide}() \\ \text{échecs :} \\ \text{termes : } \text{premier}(\text{ajouter}(u, f)) = \text{premier}(f) \end{cases} \qquad (10)$$

Etant donné une algèbre, un candidat est valide si pour toute instanciation telle que les formules des champs "contraintes", et "préconditions" s'évaluent sur l'algèbre à T et celles du champ "échecs" s'évaluent à F, celles du champ "termes" s'évaluent à T.

Il faut de plus, vu la notion intuitive

de cas d'échec proposée en [Gutt 80], que, pour toute instanciation telle que les formules des champs "contraintes" et "préconditions" s'évaluent à T, et celles du champ "échecs" s'évaluent autrement qu'à F (provoquant éventuellement une erreur), celles du champ "termes" soient telles que leur évaluation sur l'algèbre provoque une erreur.

8.3.4. Candidat et instanciation

Il nous faut prendre garde au fait suivant. Les termes instanciant les variables d'un candidat peuvent eux-mêmes introduire des préconditions ou cas d'échec supplémentaire.

Il nous faudra donc mettre à jour le candidat, plus précisément les champs "préconditions" et "échecs" tout au long du processus d'instanciation esquissé en 8.2.3.

Considérons par exemple le candidat (10). Instancions la variable f par une opération op (g), soumise à une précondition ϕ (g).

Le résultat sera le candidat suivant :

$$
\left\{
\begin{array}{ll}
\text{contraintes:} & \neg\ op(g) = file\_vide\ () \\
\text{préconditions:} & \neg\ op(g) = file\_vide\ () \\
 & \neg\ ajouter(n, op(g)) = file\_vide\ () \\
 & \phi(g) \\
\text{cas d'échecs:} & \\
\text{termes :} & premier\ (ajouter\ (n, op(g))) = \\
 & \qquad premier\ (op(g))
\end{array}
\right.
\qquad (11)
$$

8.3.5. Conclusion

Dans cette partie, nous avons décrit la notion de candidat. Cette notion joue un rôle fondamental dans le fonctionnement de notre outil. Elle permet de rendre compte du comportement spécifié par des structures de contrôle telles que les "si_alors_sinon", les préconditions, les cas d'échec ; peuvent aussi être considérées grâce à cette notion des spécifications conditionnelles.

Cette notion est essentiellement heuristique, liée aux techniques d'implantation que nous avons utilisées plus qu'à une théorie rigoureuse, puisqu'il n'existe pas, à notre connaissance de sémantique communément admise pour les notions de précondition et de cas d'échec. C'est, plus généralement, le problème de la spécification des erreurs qui est posé ; il est actuellement l'objet de nombreuses recherches.

Nous avons essayé, quant à nous, de donner quelques justifications théoriques partielles de nos choix, en 7.1.4. notamment.

8.4. Détails d'implantation

Dans cette partie, nous présentons quelques détails de l'implantation de notre outil.

Nous décrivons en particulier les principales fonctions qui le composent, et les structures de donnée utilisées.

Des listages des principaux points sont fournis en annexe 4.

Notre outil est réalisé en MACLISP, dialecte du langage LISP, et implanté sur le système MULTICS de l'INRIA, à Rocquencourt, France.

L'outil en lui-même représente environ 1000 lignes de LISP, et est divisé en plusieurs modules indépendants :

- un module utilitaire, assurant l'interface avec la description de la spécification et de l'algèbre
- un module de construction produisant les candidats associés à un axiome
- un module d'instanciation
- un module décrivant la stratégie de test
- un module assurant le test proprement dit, et son interprétation.

La description de la spécification et de l'algèbre est pour l'instant écrite directement en MACLISP. Les structures de donnée utilisée sont cependant assez simples pour que l'on puisse en envisager la production automatique à partir de descriptions plus intuitives.

Dans le cas de la présentation utilisée au chapitre 7, File (Entier), la description représente environ 100 lignes

Notre implantation a plus pour but d'étudier les problèmes liés à la réalisation d'un tel outil, que de produire un outil efficace et irréprochable. Nous avons donc utilisé un style de programmation très sûr, "fonctionnel" et modulaire, mais parfois peu efficace

8.4.1. De la spécification

La spécification sous test est implantée sous une forme analogue à une "structure" PL/1.

Une spécification est donc une liste dont les sous-listes sont de l'une des deux formes suivantes.

- formule ou définition
- (nom_du_champ (sous_liste_1 ... sous_liste_n))

La structure de donnée obtenue est très souple, très sûre, mais peu efficace car les recherches y sont très longues ; celle-ci sont analogues au comportement de la fonction LISP selectq.

Un exemple d'une telle structure de donnée, correspondant à la spécification File (Entier) utilisée au chapitre 7 est proposé à l'annexe 4. Le lecteur pourra constater que cette structure est très proche de l'écriture "naturelle" d'une telle spécification; la production automatique d'une telle structure de donnée à partir de cette dernière serait donc tout à fait envisageable à l'aide d'un système d'analyse syntaxique et de traitement d'attributs sémantiques (SYNTAX, DELTA, développés à l'INRIA par

P. BOULLIER et B. LORHO, par exemple).

Remarquons en particulier la définition des opérations, de la forme

```
( nom    (domaine_1 .. domaine_n)
         codomaine
         (exception_1 ... exception_m))
```

où nom est le nom de la fonction
domaine_i le nom du $i^{ème}$ domaine
codomaine le nom du codomaine
exception_i le mot_clé "Preconditions" ou "Echecs" suivant que l'opération est soumise à une précondition ou/et à un cas d'échec.

Les opérations sont réparties en 5 classes, conformément à 8.2.1. et 8.2.2. : "Croissantes", "Décroissantes", "Constantes", "Etales" et "Métas". L'ordre n'a pas d'importance.

Remarquons aussi la forme d'un axiome,

```
(nom    formule
        (exception_1  (nom_1 ... nom_n_1))
        (exception_p  (nom_1 ... nom_n_p)))
```

où nom est le nom de l'axiome,
formule est la liste représentant l'axiome lui-même (cf. 8.4.2.)
exception_i est le mot_clé "Préconditions" ou "Echecs", et nom_j est le nom d'une opération

apparaissant dans l'axiome, et soumise à l'exception considérée.

Nous avons ajouté un dernier champ pour faciliter les processus de construction de candidat (cf. 8.4.4.). Ils peuvent être de toute façon facilement déduits d'une analyse syntaxique de la spécification.

La liste implantant la spécification est la valeur de l'atome TYPES.

La hierarchie entre les types est définie par la valeur de l'atome ORDRE.

8.4.2. De l'algèbre

Les supports de l'algèbre peuvent en général être considérés comme des ensembles de listes, d'atomes et de chaînes de caractères (implantées en MACLISP).

Une opération de l'algèbre n'est alors rien d'autre qu'une "EXPR" (fonction LISP évaluant ses arguments). L'évaluation d'un terme est obtenu alors directement par l'évaluation (au sens LISP) de la forme LISP correspondante.

Par exemple au terme

ajouter (0, ajouter (3, file_vide ()))

correspondra la forme LISP

(ajouter (\0) (ajouter (\3) (file_vide)))

Si, par ailleurs, l'on a défini de manière naturelle le domaine des File par celui des listes LISP formés d'atomes, l'on pourra définir "ajouter" et "file_vide", "0" et "3" ainsi :

```
(defun ajouter (n f) (cons n f))
(defun file_vide () nil)
(defun \0 () 0)
(defun \3 () 3)
```

(le "\" fait du 0 et 3 des atomes au lieu de nombres).

La remarque de cette analogie extrêmement puissante entre terme abstrait et forme LISP nous a été proposée par Ph. DESCHAMP [Desc 80]. Elle est à la base de l'implantation de cet outil.

Une formule n'est donc rien d'autre qu'une forme LISP, contenant les fonctions prédéfinies EGAL, SAS (si_alors_sinon) etc.

MACLISP permet un traitement des erreurs extrêmement efficace et précis, par l'intermédiaire des fonctions error et errset. La première provoque une erreur, la seconde la récupère. Elles permettent d'écrire des opérations effectuant un "contrôle de type" sur leurs arguments.

Les méta-opérations sont implantées de manière analogue aux opérations constantes. Par exemple Entier_+ :

```
(defun Entier_+ () ( 1+ (random 10.)))
```

L'évaluation de (Entier_+) rend un entier positif aléatoire entre 1 et 11

La définition complète de l'algèbre sous test est proposée en annexe 4

8.4.3. Des candidats

La structure de donnée associée à la notion de candidat est très proche de la théorie développée en 8.3.

Un candidat est une "structure" au sens de 8.4.1., comportant trois champs : "Exigences", "Variables", "Termes."

Le champ "Exigences" est de la forme

```
(Exigences
   ( (Preconditions ( formule ... formule))
     (Echecs ( formule ... formule))
     (Contraintes ( formule ... formule))))
```

Le champ "Variables" est de la forme

```
(Variables ( vartypee_1 ... vartypee_n))
```

Le champ "Termes" est de la forme

```
(Termes ( (formule_1) ... (formule_m)))
```

Une vartypee est une variable typee, de la forme

(nom type distinction)

où nom est le nom de la variable,
type son type, et
distinction un nombre ou nil (ou rien).

Si la distinction est un nombre, la variable est du type d'intérêt, et ce nombre est la

hauteur maximale d'un terme devant instancier cette variable (et donc supérieur ou égal à 1)

Si la distinction est nil, ou si elle est absente, la variable est d'un type inférieur.

Si le champ "Variables" est de la forme

(Variables ())

le candidat est totalement instancié. Sinon, les variables y figurant sont celles restant à instancier

8.4.4. De la construction

La fonction

construire_candidat

construit, à partir d'un axiome d'un certain type les candidats correspondants.

Elle est appelée par la forme suivante
(construire_candidat axiome
nom_type
hauteur_max)

où hauteur_max est l'indice du test produit.

Cette fonction construit en particulier les champs "Préconditions" et "Echecs" du candidat selon la méthode esquissée en 8.3. La réduction d'un axiome conditionnel en sous-axiomes n'est cependant pas encore implantée, mais ne pose pas de problème particulier.

Cette fonction ne devrait donc pas être appliquée à un axiome contenant la fonction SAS (si_alors_sinon) sous peine d'obtenir un candidat incorrect.

8.4.5. De l'instanciation

La fonction

instancier

instancie une vartypee donnée d'un certain candidat par une certaine opération, mettant à jour les champs "Preconditions" et "Echecs" du candidat obtenu (cf. 8.3.4.)

Cette fonction est appelée sous la forme suivante

(instancier candidat
vartypee
opération
message)

candidat et vartypée ont été décrit en 8.4.3., opération en 8.4.1.

message est soit un nombre, soit le mot-clé "Ne_pas_distinguer". Dans le premier cas, les nouvelles variables introduites du même type que vartypee seront affectée de ce nombre pour distinction.

Ces nouvelles variables, pour éviter tout conflit de nom, sont produites par un appel de la fonction gensym.

La fonction instancier n'est utilisée que dans le cas des opérations pures (cf. 8.2.1.)

8.4.6. De l'évaluation

La fonction

evaluer

evalue sur l'algèbre considérée un candidat totalement instancié d'un certain type.

Elle est appelée sous la forme suivante

(evaluer candidat
nom_type)

Cette évaluation n'est faite que lorsque toutes les variables du candidat ont été instanciées. Le champ "Variables" est alors de la forme

(Variables ())

La légitimité du candidat (cf. 8.5.) est d'abord examinée.

Les formules du champ "Contraintes" doivent toutes s'évaluer à true, sans erreur.

Les formules du champ "Préconditions" aussi

Si celles du champ "Echecs" s'évaluent toutes à nil, sans provoquer d'erreur, alors le candidat est légitime, et les formules du champ "Termes" doivent s'évaluer à true, sans erreur, <u>sinon la spécification est incorrecte</u>

Si celles du champ "Echecs" ne s'évaluent pas toutes à nil (ou provoquent des erreurs), celles du champ "Termes" <u>doivent</u> provoquer

une erreur à l'évaluation, <u>sinon la spécification est incorrecte</u>.

Dans les autres cas, l'évaluation du champ "Termes" n'a qu'une valeur indicative.

L'évaluation d'une formule, par la remarque du 8.4.2., se ramène à l'évaluation (au sens LISP) de la forme

(erset (eval formule))

qui vaut nil si une erreur s'est produite pendant l'évaluation, et la liste formée de la valeur résultante sinon.

8.4.7. De la stratégie

La fonction

strategie_1_candidat

réalise la stratégie d'instanciation et d'évaluation pour un candidat d'un certain type.

Sa valeur est la liste des candidats instanciés produits à partir du candidat donné.

Elle est appelée sous la forme

(strategie_1_candidat
candidat
nom_type).

Le texte de cette fonction, ainsi que des fonctions annexes qu'elle utilise est proposé en annexe 4.

Comme le lecteur pourra le constater, cette fonction est hautement récursive, par l'intermédiaire de la fonction strategie_candidats.

strategie_1_candidat cherche d'abord la première vartypee du candidat. S'il n'en existe plus, c'est que le candidat est totalement instancié, et elle appelle la fonction d'évaluation (cf. 8.4.6.)

Sinon, elle examine cette vartypee en appelant stratégie_1_candidat_1.

Si la distinction vaut 1, soit x cette variable. Selon 8.2.3 et 8.2.4., x doit être remplacée soit par une constante du type d'intérêt, soit par une méta-opération, soit par une opération croissante de la forme $\varphi(y_1 \dots y_n)$, $y_1 \dots y_n$ étant des variables de type inférieur.

Si la distinction est nil, c'est une variable de type inférieur, et elle est instanciée soit par une constante, soit par une méta-opération.

Si la distinction est strictement supérieure à 1, l'on retrouve le cas 1 avec en plus une opération étale $\varphi(x_1 \dots x_n\ y_1 \dots y_m)$, les x, variables du type d'intérêt, étant affectés de la distinction égale à celle de la variable diminuée d'une unité.

Donc, à chaque appel, la distinction des variables introduites diminue, ou une variable est éliminée. L'algorithme termine donc bien

Les instanciations par des opérations pures se font par l'appel de la fonction instancier.

Par contre, celles concernant les méta-opérations procèdent différemment.

C'est en effet un élément "aléatoire" de l'algèbre qui est considéré, et non une opération qui ne sera évaluée que l'instanciation terminée. Ces élément aléatoire est la <u>valeur</u> de la méta-opération.

Examinons le cas de Entier_+

Supposons que, à cet instant,

Entier_+ () = 4, ou plutôt

(Entier_+) = 4.

Nous devons substituer "4" à la variable y du candidat. Mais il faut que cette valeur soit préservée lors du processus d'évaluation terminal. Aussi, bien que ce ne soit ici pas strictement nécessaire puisque la valeur de 4 est 4 en LISP, il nous faut substituer

(quote 4)

à la variable y.

Plus généralement, la méta-opération est de la forme

(nom () codomaine);

il faut substituer

```
(list 'quote
      (eval (car opération)))
```

à la variable.

C'est le rôle de la fonction instancier_candidat_1_Méta, dont le texte est en annexe 4.

8.4.8. Conclusion

Nous avons dans cette partie, essayé de décrire précisément certaines parties de l'implantation de notre outil.

Comme nous l'avons dit, cette implantation est purement expérimentale et aucun problème d'efficacité n'y est réellement abordé.

Il nous semble que ces détails d'implantation montre que la réalisation d'un tel outil est relativement simple.

Il est en particulier possible de prendre en compte, de manière intuitivement satisfaisante, les problèmes non encore théoriquement résolus posés par les préconditions et cas d'échec, ainsi que par les "si_alors_sinon"

Le langage LISP est à notre avis particulièrement bien adapté à une telle implantation essentiellement récursive. Il permet d'autre part de manipuler des structures de donnée souples, pratiquement non limitées et très tolérantes. La possibilité de traitement d'erreur est fondamentale. L'utilisation d'une version compilée de l'implantation permettrait sans doute une efficacité d'exécution comparable à celle

obtenue par l'utilisation de langages impératifs (PL/1 notamment).

Enfin, signalons que le dialecte MACLISP disponible sur le système MULTICS est particulièrement agréable à utiliser, par la possibilité d'interactions extrêmement fines avec le système depuis l'interpréteur. Un compilateur particulièrement performant est aussi un atout de choix pour ce dialecte. L'éditeur "intelligent" EMACS admet un mode LISP très agréable, quoique certaines subtilités soient dangereuses pour la survie de l'éditeur (mode LDEBUG particulièrement). Bref, seul manque, pour utiliser agréablement les immenses possibilités de MACLISP, un bon manuel d'utilisation.

8.5. Conclusion

Le chapitre était consacré à une généralisation de l'exemple présenté au chapitre 7. Cette généralisation concerne la validation de spécifications abstraites sur des algèbres au moyen du test.

La réalisation effective d'un outil de validation automatique, décrit notamment en 8.4., et sa relative simplicité (1000 lignes de LISP) montre qu'à l'évidence un tel projet n'est plus de l'ordre du rêve, mais est à portée de main.

Etant donné les descriptions d'une spécification abstraite et d'une algèbre, notre outil est capable de réaliser, entièrement automatiquement, le processus de test d'un axiome de cette spécification sur cette algèbre, et ce, à un niveau choisi par l'utilisateur. A celui-ci de choisir, en fonction du coût admissible, le niveau le plus rentable ; un niveau infini permettrait éventuellement de conclure avec certitude à la correction de l'algèbre par rapport à sa spécification.

Il nous semble fondamental de remarquer que la réalisation de cet outil de validation automatique n'a pu être possible que par l'effort théorique important qui le sous-tend. Cet effort nous permet en particulier d'énoncer précisément les conditions d'utilisation de cet outil (cf. 8.1.2., 8.1.4., 8.1.5.).

Il n'est, en particulier, pas applicable à des spécifications analogues à celle de Ensemble (Entier).

Pour pouvoir traiter ce type de spécification, il nous faudrait des heuristiques de transformation de représentation construites sur des bases théoriques rigoureuses. De telles heuristiques ont été esquissées en [Bidoit 81]. Leur application à notre problème mériterait sans doute une étude approfondie (cf. 9.3.)

Signalons enfin l'application extrêmement prometteuse de notre outil à la validation de compilateurs proposée par [Gaudel 80], qui ramène la compilation à une fonction de représentation entre spécifications abstraites.

Chapitre 9 : Conclusion

"Les sciences expérimentales n'ont fait de progrès qu'à partir du moment où des mathématiciens s'y sont intéressés"

Il en a été ainsi pour la physique il y a deux siècles, pour la chimie il y a un siècle, pour l'économie il y a trente ans, pour la biologie il y a dix ans.

De nombreuses branches de l'informatique ont depuis longtemps été renouvelées, vivifiées par l'introduction d'un soubassement mathématique rigoureux. La notion de test, sans doute rejetée dans l'ombre par celle de preuve trop proche, avait jusqu'à présent été négligée

Nous souhaitons que ce travail, prolongement et approfondissement des propositions formulées pour la première fois en 1975 par J.B. GOODENOUGH & S.L. GERHART, soit, pour la notion de test, l'amorce d'un tel renouveau.

Nous en résumons ici les principaux résultats, et les problèmes qu'ils dévoilent.

9.1. De la modélisation

La force de la modélisation de la notion de processus de test réside dans sa globalité. Nos prédécesseurs ne s'étaient intéressés qu'à la notion de test, isolée artificiellement de celle de processus de test. Ils étaient donc dans l'incapacité de rendre compte correctement du fait que si un test est, ou n'est pas, réussi, ce résultat n'induit qu'une présomption de validité pour la propriété sous test. Il était en effet impossible de mêler l'esprit manichéen qui règle la réussite ou l'échec d'un test, et l'esprit qualitatif qui guide la conclusion du test.

Nous avons résolu ce problème en distinguant plusieurs phases dans un processus de test, celle de construction relevant de l'esprit manichéen de la logique du premier ordre, celle d'application relevant du principe de couplage. Celui-ci essaye de rendre compte du fait suivant: la qualité de la conclusion du test est liée à la qualité de la validité des hypothèses sous-jacentes.

Afin d'obtenir une modélisation manipulable mathématiquement, nous avons dû distinguer

un niveau concret et un niveau abstrait, ce qu'avaient en fait implicitement réalisé nos prédécesseurs. La notion de processus comprend donc une phase de représentation, de traduction, d'abstraction difficile à maîtriser. Par contre, la phase de construction est entièrement conduite au niveau abstrait, ce qui permet l'élaboration d'un formalisme rigoureux pour les notions de contexte de test, de jeu de tests et de restriction de contexte.

9.2. Du développement mathématique

Nos prédécesseurs n'avaient pu aborder réellement ce problème à cause de l'insuffisance de leur modélisation, de son caractère trop concret.

Nous avons, pour notre part, tenté de choisir avec soin et avec un esprit critique les éléments constituant notre modèle, et notamment le formalisme mathématique sous-jacent. Celui de la logique du premier ordre égalitaire nous a paru un compromis raisonnable. Il est par nature proche de l'intuition que nous avons du problème, mais aussi assez classique, assez "rodé" pour que nous puissions bénéficier d'une grande expérience technique, d'une "tradition spirituelle" liée à cette outil.

Il nous a semblé important de rester, dans toute la mesure du possible, fidèle, dans le traitement mathématique de notre modèle, à l'esprit, à la "nature" de la logique du premier ordre. Ce choix a pu être parfois lourd à assumer, notamment par la complexité des définitions auxquelles il nous a conduit. Cependant, il nous semble qu'il se révèle à terme largement rentable, comme le montrent les théorèmes remarquables de stabilité, d'équivalence, de localisa-

tion, qui permettant de décrire et de justifier une méthode rigoureuse de construction de jeux de tests. En fait, la subtilité de la distinction entre propriétés formelles et logiques nous donne une puissance d'expression largement supérieure à celle proposée par l'intuition première.

9.3. Des applications pratiques

Les applications de notre travail que nous avons décrites n'ont pas tant pour but de montrer "à quoi sert" notre travail que de valider notre modèle en démontrant qu'il est capable de rendre compte de l'intuition qui en est la base et l'inspiration.

Notre formalisme nous a permis de développer de nombreux outils rigoureux permettant de contrôler, de soutenir, d'orienter la construction d'un "bon" jeu de tests pour un problème donné. Nous avons pu formuler des conditions (suffisantes) précises pour que ces outils soient applicables et efficaces. Nous sommes capable de produire, par l'utilisation cohérente de ces outils, un jeu de tests au moins aussi satisfaisant que celui qu'aurait proposé empiriquement un programmeur averti, et ce par des méthodes relativement "naturelles". Mais, en plus, le fait de pouvoir s'appuyer à tout instant sur une base théorique rigoureuse nous permet de définir exactement à quelles conditions, et selon quels critères ce jeu de tests est "satisfaisant". Nous sommes en particulier capable de distinguer précisément la notion de test de celle d'"échantillonage aléatoire" du domaine, de préciser la qualité

d'une optimisation, de définir l'équivalence de deux jeux de tests.

S'il se trouve que les types abstraits algébriques offrent un terrain d'application privilégié à notre travail, cela est dû essentiellement aux liens de parenté étroits existant entre ce formalisme et la logique du premier ordre. Notre sentiment est que le formalisme que nous avons proposé devrait pouvoir intégrer et utiliser de manière profitable l'expérience acquise par les théoriciens s'intéressant aux types abstraits algébriques.

A titre d'exemple, citons les <u>méthodes de transformation de présentations</u>. Dans quelle mesure influent-elles sur le test des axiomes d'une présentation sur une algèbre donnée?

En fait, plus fondamentalement, nous avons toujours considéré A, la théorie sous test, comme une donnée immuable. Qu'arrive-t-il si cette donnée devient une variable, comme les hypothèses H ou la famille de tests $(T_n)_{n \in \mathbb{N}}$?

L'examen de ce problème dans l'"esprit" des types abstraits algébriques nous semble prometteur.

9.4. De la notion de qualité

Nous terminons cette thèse en laissant en fait à peine débroussaillé l'un des problèmes essentiels posés par la notion de test (plus précisément de processus de test).

Ce problème est celui de la qualité d'un processus. Plus précisément, comment quantifier, ou même simplement estimer, la confiance que l'on peut avoir en la conclusion d'un processus de test ?

La difficulté est que la notion de qualité est typiquement probabiliste, puisqu'elle s'intéresse au comportement "moyen" d'un système. Or, par le choix de la logique du premier ordre, nous sommes très éloigné des techniques probabilistes; elles nous paraissent, quoique très naturelles, difficiles à mettre en oeuvre ici, à moins d'une refonte complète de notre travail.

Il nous faut donc trouver autre chose, et cette thèse nous fournit quelques indications. Cette qualité "intrinsèque", "globale" n'est fonction que de celle des restrictions conservatives opérées lors de la phase de construction, et du choix du test lors de la phase d'application.

Le second problème est lié à l'étude

des chaînes de théories, et semble très technique. Par les équivalences que nous avons définies, nous savons au moins que l'indice du test choisi ne joue aucun rôle intrinsèque dans la qualité du processus.

Le premier problème revient pratiquement à <u>évaluer la validité des hypothèses</u> de régularité et d'uniformité sur les familles de structures des contextes considérés. Il nous faut donc définir plus précisément ce qu'est un sous-domaine uniforme. Une piste peut-être intéressante est proposée par les types abstraits algébriques, considérés comme systèmes de réécriture; deux termes sont intuitivement uniformément traités par les opérations du type si les mêmes réécritures (à équivalence près) sont possibles sur les deux termes; autrement dit, les deux termes admettent des ensembles de calculs (éventuellement infinis) équivalents. Par rapport au type Entier, $\mathbb{Z}$ devrait ainsi être divisé en trois sous-domaines : $\{0\}$, $\{succ^i(0), i>0\}$, $\{pred^i(0), i>0\}$.

Ce problème semble difficile, et ne peut être abordé qu'avec l'aide d'un formalisme élaboré constamment soumis à la critique de l'intuition. Il constitue cependant, à notre avis, la suite logique de notre thèse.

9.5. Envoi

"We know less about the theory of testing
that we do often,
than about the theory of proving
that we do seldom.
This paper is a step toward redressing
this imbalance."

Toward a theory of test data selection
J.B. GOODENOUGH & S.L. GERHART
1975

Historique de la notion de test

Nous reproduisons dans les pages qui suivent un extrait du rapport intitulé Validation de programmes par test : théorie et pratique [Boug 82].

Ce rapport de synthèse consacré au problème de la validation de programmes, et plus précisément de compilateurs, a été le point de départ de notre thèse.

Au chapitre 2, après avoir introduit la notion de validation, celle d'erreur et celle de fiabilité, nous présentons les différents essais de formalisation, de modélisation théorique de ces notions proposés depuis une dizaine d'années.

Les références entre parenthèses renvoient à la bibliographie présentée à l'annexe II.

CHAPITRE 2 : FORMALISATION DE LA NOTION DE TEST

Dans le chapitre précédent, nous avons étudié ce que l'on attend d'un programme - une certaine fiabilité - et ce qu'on y trouve généralement - des erreurs de types très divers. Dans ce chapitre, nous essayons de formaliser ces notions. Nous définissons ce que nous appelons correction d'un programme, et précisons la notion d'erreur en proposant une ébauche de modèle. A partir de ces notions, nous définissons ce qu'est un test, ou plus généralement un critère de test. Nous décrivons les différentes propriétés que l'on peut attendre d'eux : fiabilité, validité, idéalité, ainsi que leur diverses variantes proposées dans la littérature.

2.1 Présupposés et hypothèses

On s'intéresse ici à un programme (une suite d'instructions écrites dans un langage de haut niveau, de type Pascal ou PL/1, pour fixer les idées), placé dans un environnement réel : ce que nous appelons programme est donc la suite de caractères proposée à la machine par l'intermédiaire de son organe d'entrée ; ce que nous appelons résultat est la suite de caractères exprimée par la machine sur son organe de sortie. Remarquons que, dans cette optique, les notions de programme et de donnée ne sont pas fondamentalement distinctes, car ce n'est qu'une même suite de caractères proposée à la machine.

L'utilisateur est censé ne pouvoir accéder qu'à la forme externe du programme (suite formelle de caractères comprenant instructions et données). L'environnement interne de la machine (compilation, génération de code, gestion de la mémoire, matériel) lui est inconnu. On fait donc en général (implicitement) l'hypothèse que cet environnement est parfait, en un certain sens. Posons simplement que tous les comportements externes du système, exécutant le programme sur les données, ne correspondant pas aux spécifications, peuvent être modifiés et corrigés par une modification du programme, en tant que suite d'instructions de haut niveau. Plus simplement, on fait l'hypothèse que l'environnement gère "parfaitement" le programme (instruction & données), sans y introduire d'erreur. L'environnement est transparent pour les spécifications.

On fera également l'hypothèse que l'on possède un "oracle", permettant de décider si une chaîne de caractères exprimée répond aux spécifications du problème pour les données considérées ; en d'autres termes, si le système répond de manière satisfaisante ou non. Il faut remarquer ici que la réponse

du système peut être de formes très diverses. Ce peut être bien sûr la chaîne de caractères "produite par le programme", mais aussi bien des messages d'erreur provenant du compilateur, de l'éditeur de liens, ou du système proprement dit. Un bouclage infini (et donc l'absence totale de sortie) peut être considéré, du reste, comme une réponse à part entière. Remarquons, pour finir, que ces hypothèses sont extrêmement restrictives, et au premier chef celle concernant l'existence d'un oracle (cf. 2.2).

2.2 Notion de correction d'un programme

2.2.1 Définition (C1)

Soit P un programme (suite d'instructions) et D son domaine d'entrée.

En règle générale, on a $D=A^*$, où A est l'alphabet utilisé par l'organe d'entrée du système. En général, aussi, le domaine de sortie de P est contenu dans D.

Une spécification de P est une application F

$$F : D \rightarrow \mathcal{P}(D \cup \{\Omega\})$$

qui, à toute donnée associe l'ensemble des réponses acceptables. L'oracle, que nous noterons OK, est donc un prédicat sur D :

$$OK : D \rightarrow \{vrai, faux\} \quad OK(d) = (P(d) \in F(d))$$

où $P : D \rightarrow D \cup \{\Omega\}$ est l'application qui a toute donnée fait correspondre la réponse du système dirigé par le programme P. P est correct si : $\forall d \in D, OK(d)$

Plusieurs remarques s'imposent.

Cette notion de correction du programme P n'est définie que comme rapport entre entrées et sorties. Elle ne tient aucun compte de la structure interne de P, et, en particulier, ne rend absolument pas compte des performances de P. C'est une approche de type "boîte noire".

Il faut prendre conscience que dans la pratique, l'on n'applique un programme qu'à une très petite partie du domaine d'entrée théorique. Il y a une notion de bon sens qui n'est pas prise en compte par cette définition. Qui s'est jamais demandé quelle devait être la réponse du système devant 2 + "abc" ?

2.2.2 Le prédicat OK : rêve ou réalité ? (C3)

Je m'inspire ici largement de l'article de E.J. WEYUKER (C3). Remarquons tout d'abord que cette notion d'oracle, i.e. la possibilité effective de décider si la réponse produite par un système répondant à une stimulation

connue est correcte ou non, est fondamentale dans le formalisme qui précède. D'autre part, les programmes envisagés dans la littérature sont relativement simples, et leur domaine assez bien défini pour que l'on puisse déterminer un tel oracle.

Dans la pratique, cependant, l'on doit se contenter d'une présomption de correction. Si l'on ne peut que rarement affirmer qu'une réponse est correcte par rapport à une spécification, l'on peut souvent décider si elle est plausible ou non. En d'autres termes, si l'on ne peut jamais décider qu'une réponse est correcte, il y a de nombreux cas où l'on peut affirmer qu'elle est incorrecte. Si un programme qui doit calculer PI au dix-millionième, répond 3,1415932, il est peut-être correct. S'il répond 4,3121521, il contient sûrement une erreur.

Remarquons que cette notion de "réponse plausible" est extrêmement floue. Est-ce que 5 est une valeur plausible de la millième décimale de PI ? D'autre part, le premier travail dans la réalisation d'un programme consiste à passer d'une spécification générale à une spécification algorithmique ; dans notre cas, passer de "trouver la millième décimale de PI" à l'algorithme mathématique qui donne ce résultat. Si l'erreur du programme est en fait la traduction d'une erreur dans l'algorithme, et si, pour déclarer une valeur plausible, l'on exécute cet algorithme de manière approchée, le problème n'avance pas.

Un autre cas, courant, est celui où le volume d'information produit par le programme est trop grand pour être vérifié (même si, ponctuellement cette vérification est triviale). On déclare alors la réponse plausible sur la vérification d'un échantillonnage présumé particulièrement sensible. Mais là encore, les conclusions restent vagues.

Dès lors, comment décider de la correction d'un programme par rapport à ses spécifications ? Quel oracle choisir ? WEYUKER en propose trois. La première méthode consiste dans l'écriture, à partir des spécifications, de deux algorithmes et donc deux programmes indépendants, P et P . Si pour un stimulus donné, les deux programmes ont le même comportement externe, la réponse sera dite correcte.

$$\forall d \in D \quad OK(d) = [P_1(d) \equiv P_2(d)].$$

Nous retrouvons cette méthode dans les stratégies de test du matériel, ainsi que dans les problèmes de validation de compilateurs (compilateur de référence, machines virtuelles, ...).

Une seconde méthode consiste à modifier légèrement P (altérer la valeur des constantes, par exemple) pour se ramener à une spécification plus facile à manipuler ; par exemple, si P doit calculer la millième décimale de PI, il sera facile de le modifier ponctuellement pour lui faire calculer la troisième décimale : $\forall d \in D \quad OK(d) = OK'(d)$ OK' étant l'oracle associé au programme modifié.

Enfin, la troisième méthode consiste à calculer, par un algorithme indépendant de celui utilisé par le programme, un encadrement aussi réduit que possible de la solution :

$$\forall d \in D \quad OK(d) = [P(d) \in [inf(d), sup(d)]]$$

En conclusion, il faut signaler à nouveau que sans "oracle", la notion même de test est vide de sens. En effet, à quoi sert une stratégie de test, si fiable soit-elle, si dans la pratique l'on est incapable de décider si un test a été passé avec succès ou non ? Bien que ce problème ne soit pas abordé plus loin, tout ce qui sera écrit ici sur l'efficacité des méthodes de test est bien sûr dépendant de la qualité de l'oracle dont on dispose.

2.3 Notion d'erreur (E1)

2.3.1 Définition

Nous reprendrons ici largement l'article de HOWDEN (E1).

Il propose la définition suivante :

Une erreur dans un programme P est la différence (syntaxique) entre P et un (hypothétique) programme réalisant la spécification F de P. Nous noterons ce programme idéal P*. En d'autre terme, HOWDEN définit une erreur comme ce qu'il faut changer dans le texte de P pour obtenir un programme correct.

Cette notion est ambiguë. En effet, il y a souvent plusieurs modifications syntaxiques possibles permettant d'assurer la correction de P. Dans le cas où la spécification de P n'est pas univoque (plusieurs comportements sont acceptés comme corrects), ces variations syntaxiques ne sont pas forcément équivalentes sémantiquement. D'autre part, la notion d'unité d'erreur n'est pas précisée. Comme le remarque HOWDEN, il est parfois intuitif de décrire un programme contenant plusieurs instructions incorrectes comme ne comportant qu'une seule erreur. En général, cependant, on considérera que chaque instruction incorrecte introduit une erreur propre.

Cette notion d'erreur est très imparfaite. Elle rend cependant bien compte de l'idée intuitive que nous avons d'un programme "presque correct" telle que la précise l'étude de ENDRES (B12) : dans un voisinage syntaxique "petit" de P, on peut trouver un programme correct. Cet a priori est justifié par le fait que la validation par test est effectuée généralement en vue de la certification, c'est-à-dire alors que le programme est considéré intuitivement comme "acceptable".

2.3.2 Indépendance

Si P contient les erreurs (différences syntaxiques d'avec $P*$) $E_1,\ldots,E_n$ on notera $P*E_i$ le programme obtenu en perturbant P par la modification syntaxique E_i. On a donc $P*E_1,\ldots,E_n = P$

On dira que l'erreur E_i est indépendante des autres si celles-ci ne compensent pas les incorrections qu'elle introduit.

$$\forall d \in D \left[P * E_i(d) \neq P * (d) \Longrightarrow P(d) \neq P * (d)\right]$$

On peut faire ici les mêmes remarques qu'au paragraphe précédent. Cette définition est bien adaptée au cas d'un grand programme "presque correct" où la modification de quelques instructions dispersées suffirait pour assurer la correction.

2.3.3 Modèles d'erreur

Pour tenter de lever l'ambiguïté des différentes notions d'erreur proposées dans la littérature, de nombreux auteurs ont essayé, à partir des classifications empiriques exposées dans le chapitre précédent, de donner un modèle d'erreur. On trouve cette idée chez HOWDEN (E1) reprise chez WHITE et COHEN (D21). Elle est citée dans (C1).

L'idée de base est la suivante : l'on peut partitionner le domaine d'entrée D de telle manière que le programme traite de manière "uniforme", en un sens à préciser, deux données de la même classe. Nous appellerons désormais une telle classe <u>sous-domaine d'uniformité</u>. L'idée de HOWDEN est de ne considérer pour ce partitionnement que le texte formel du programme, une classe correspondant à l'ensemble des données qui suivent un même "chemin" à travers le programme. On peut raffiner la méthode, dans le cas des stratégies de test, en intersectant la partition déduite de la structure de contrôle du programme avec celle déduite des spécifications (C1), (D4).

WHITE, suivant HOWDEN, propose donc d'associer à chaque classe la suite d'instructions correspondant au chemin considéré. On en déduit donc une "représentation canonique" du programme comme un ensemble de couples

$$\{(D_1, f_1), (D_2, f_2), \ldots\}$$

avec D_i : ième s/domaine d'uniformité,

F_i : fonction calculée par la suite d'instructions rencontrées par les données de D .

remarquons que, bien que définie par D_i, f_i est une application sur D, à valeur dans $D \cup \{\Omega\}$.

Il faut tenir compte, d'autre part, dans la suite d'instructions considérée, des "effets de bord" possibles des prédicats dont l'évaluation peut modifier les valeurs des variables.

HOWDEN définit ainsi trois types d'erreurs :

2.3.3.1 Erreur de frontière

L'un des sous-domaines d'uniformité D_i est incorrect par rapport aux sous-domaines correspondant de P*, mais la fonction f_i calculée par la suite d'instructions que rencontre les données de D_i est correcte.

Par exemple :

```
...                                  ...
si x > 2 alors ...   au lieu de   si x ≥ 2 alors ...
...                                  ...
```

On peut schématiser la chose par le premier croquis de la figure 2.1 : les données situées dans la zone hachurée sont traitées incorrectement par P, en général, car il se peut que pour certaines valeurs d, $f_1(d) = f_2(d)$. Nous appelons un tel cas une <u>coïncidence accidentelle</u>.

2.3.3.2 Erreur de calcul

Dans ce cas, le partionnement de D_i en sous-domaines par P est correct (i.e. est le même que celui opéré par P*), mais l'une des fonctions f_i est incorrecte.

Par exemple :

```
...                          ...
x = x+2        au lieu de    x = x+1
...                          ...
```

Dans ce cas, les données de D_i sont traitées en général incorrectement. Il se peut, cependant, que pour certaines d'entre elles, on ait une coïncidence accidentelle entre la valeur calculée, et la valeur correcte.

2.3.3.3 Erreur de partitionnement ou de raffinement

Dans ce cas, l'un des sous-domaines D_i déterminé par P est trop grand : les données y sont calculées de manière uniforme alors que, dans le programme correct P*, D_i est encore subdivisé en sous-domaines plus petits. Grossièrement, il manque un test dans le programme.

Par exemple :

```
...                                          ...
si x > 2 alors ... is   au lieu de   si x > 2 alors
...                                  si x > 3 alors ... is
                                     sinon ...  is
                                     ...
```

Dans le deuxième schéma de la figure 2.1, les données de la partie hachurée sont traitées (en général) incorrectement.

2.3.4 Justification de ce modèle

Nous avons exposé le modèle d'erreur proposé entre autres par HOWDEN. Nous voudrions insister sur l'utilité d'un tel modèle. Il est bien évident que, dans la pratique, une erreur n'appartient jamais à une seule de ces classes. En effet, l'altération d'une instruction du programme, par exemple, modifiera en général profondément le flot de contrôle de celui-ci, et donc la subdivision du domaine d'entrée en sous-domaines d'uniformité. En général, donc, une erreur de calcul s'accompagnera d'une erreur de frontière. Mais la force d'un modèle ne réside pas dans sa conformité avec la réalité, l'exemple du modèle "stuck-at" pour les fautes en matériel est particulièrement révélateur. Ce que HOWDEN affirme, c'est seulement ceci : si l'on peut démontrer qu'une stratégie de validation détecte la présence d'une erreur de l'un de ces trois types, alors elle détectera presque sûrement la présence d'erreurs, même multiples, dans le programme.

Ce modèle n'est pas justifiable dans le cas de n'importe quel programme. Il nous semble, cependant, que, d'après les résultats du chapitre 1, il soit bien adapté aux cas de programmes en cours de validation et "quasi-correct", au sens où chaque erreur peut être corrigée par la modification d'une seule instruction.

Soulignons cependant deux faiblesses de ce modèle : d'une part, il est fondé sur la structure interne du programme, alors que la notion d'erreur est relative au comportement externe du programme plongé dans son environnement. D'autre part, les trois types d'erreurs ne sont pas comparables. Si les deux premiers sont "aisément" décélables, le troisième est beaucoup plus subtil. La plupart des stratégies de test exposées ne seront d'ailleurs pas fiables quant à lui.

2.4 Notion de test

2.4.1 Définitions : test, réussite, complétude, fiabilité, idéalité

Nous reprenons ici les définitions proposées par GOODENOUGH et GERHART dans leur article de référence (C1).

Soit P un programme, de spécification F sur un domaine D ; on appellera test un sous-ensemble fini T de D.

On dira que le test réussit si chacune des données fournit une réponse correcte.

REUSSITE (T) = $\forall t \in T$, OK(t)

La notion de stratégie est formalisée sous la forme d'un critère de sélection C. Si T est un test sélectionné par C, on dira que T est complet par rapport à C : COMPLET (T,C). Remarquons que de très nombreux tests peuvent être sélectionnés par un même critère. On peut voir C comme un ensemble de parties de D.

On dira qu'un critère (une stratégie) est fiable si tout test qu'il sélectionne aboutit au même résultat.

FIABLE (C) = $\forall$T1 T2 $\subseteq$ D,

[(COMPLET (T1,C) & COMPLET (T2,C)) ==>

(REUSSITE (T1) $\equiv$ REUSSITE (T2))]

On dira que C est valide, si toute réponse incorrecte potentielle est mise en évidence par l'un des tests sélectionnés par C.

VALIDE (C) = $\forall d \in D$

[$\neg$OK(d) ==>($\exists$ T $\subseteq$ D,

COMPLET (T,C) & $\neg$REUSSITE (T))]

On dira que C est idéal, s'il est à la fois fiable et valide. Dans ce cas, en effet, s'il y a une erreur dans P, elle sera mise en évidence par n'importe quel test sélectionné par C.

IDEAL(C) = VALIDE(C) & FIABLE(C)

2.4.2 Remarques et critiques

Il faut toujours se méfier des définitions : par exemple C = $\emptyset$ est un critère fiable. De même si C = {T} ne contient qu'un seul élément.
Si le programme est correct, tout critère est idéal, en particulier $\emptyset$.
D'autre part, tout critère est soit fiable, soit valide (C2) .

L'idéalité est, en général, extrêmement difficile à mettre en évidence. En effet, démontrer l'idéalité d'un critère revient, en partie, à prouver la correction du programme, puisqu'elle est impliquée par la réussite de n'importe quel test sélectionné. Il est donc probable que démontrer l'idéalité d'un critère soit aussi difficile que de prouver formellement la correction d'un programme, et de toute façon impraticable dans le cas de programmes réels.

L'un des défauts de ces définitions est qu'elles font intervenir le programme lui-même. Supposons donné un critère C idéal pour un programme. Si l'on modifie P, par exemple en y introduisant (involontairement) une erreur,

C ne sera en général ni fiable ni valide pour le programme ainsi altéré. On trouvera dans (C2) et (C4) une discussion plus détaillée de ce problème. Il faut noter d'ailleurs que la notion de fiabilité d'un test T développée dans (C4) et (E1) correspond, comme chacun s'y attendait, à la validité du critère C = T , et non à sa fiabilité !

2.4.3 Une proposition : notion de test révélateur d'un sous-domaine (C2)

Le principal reproche que WEYUKER & OSTRAND (C2) font au formalisme précédent concerne la quantité d'information manipulée : en effet, les définitions font intervenir le domaine D tout entier alors que dans la pratique, seule une petite partie est considérée. Il est donc inutile de vouloir tester le comportement du programme sur tout son domaine d'entrée. Par contre, il est évident qu'en général, une erreur ne pertube pas seulement une donnée, mais tout un sous-domaine. Comme il est possible, dans une certaine mesure, de délimiter a priori ces sous-domaines, il suffira d'aller tester une valeur dans chacun d'eux pour mettre en évidence toutes les erreurs potentielles. Les auteurs proposent donc ces définitions. Un critère C est révélateur pour un sous-domaine $S \subseteq D$ si tout test de C est rejeté dès qu'une donnée de S est incorrectement traitée par P.

$$\text{REVELATEUR}(C,S) = [(\exists d \in S)\ \neg OK(d)] \Longrightarrow$$
$$[(\forall T \subseteq D)\ (\text{COMPET}(T,C) \Longrightarrow \neg \text{REUSSITE}(T))]$$

Pour S = D et C = Ø, ceci est équivalent à IDEAL (C)

A ce point, tout revient donc à partionner D de manière judicieuse. L'idée qui vient immédiatement à l'esprit est d'utiliser les sous-domaines d'uniformité, dont nous avons parlé plus haut, et de prendre comme critère de test le fait de tester au moins une valeur dans chaque sous-domaine. On peut raffiner la partition à partir des spécifications et des "cas limites" qu'elles mettent en évidence afin de la rendre moins dépendante de la structure interne du programme à tester.

On peut aussi ne se restreindre qu'à la recherche d'un certain type d'erreur, ce qui conduit à la notion de sousdomaine révélateur d'une erreur E

S est révélateur pour E si $\forall d \in S\ (P *_E (d) \not\in F(d))$

ce qui signifie que la présence de E perturbe toutes les données de S. A partir de cette notion, on peut construire un test qui détectera la présence de E, en supposant que cette erreur n'est pas compensée (erreur indépendante).

2.4.4 Une autre proposition : notion de test discriminant

(D33), (C4)

Cette proposition naît de la même remarque que la précédente. La notion d'idéalité est trop forte par rapport au comportement pratique d'un programmeur. D'autre part, la notion de test révélateur reste contingente au programme lui-même : si l'on modifie celui-ci, un test révélateur pour l'ancien programme ne le restera pas en général pour le nouveau puisque les sous-domaines se trouvent bouleversés. La qualité d'un test ne doit donc pas seulement dépendre du programme lui-même, mais de tous les programmes se trouvant au voisinage de celui-ci.

Soit donc P un programme de spécification F.
On dira qu'un test T est discriminant pour P dans un voisinage V(P) si

$$\forall Q \in V(P) \quad [(\forall d \in T \quad Q(d) = P(d)) \Longrightarrow (\forall d \in D \quad Q(d) = P(d))]$$

En d'autres termes, T suffit à distinguer P de tous les programmes qui lui ressemblent.
En pratique, la condition du second membre de l'implication sera moins forte : on imposera seulement $Q \simeq P$, et la relation d'équivalence $\simeq$ sera plus large que l'équivalence sémantique stricte. Par exemple, ce pourra être seulement l'équivalence sémantique sur un sous-domaine S de D :

$$Q \simeq P \quad \text{ssi} \quad [\forall d \in S \quad Q(d) = P(d)]$$

Cette notion de test discriminant n'a d'utilité, dans notre cas, que si P est suffisamment proche de P* ($P^* \in V(P)$). Alors si T distingue P de ses voisins, et si P n'est pas correct, T distinguera P du programme correct P*. Cette hypothèse est formulée dans (D33) par T.A. BUDD etc. sous le nom de "hypothèse du programmeur compétent". Elle reprend ce que nous exprimions au paragraphe 2.3.4 par la notion de "quasi-correction" d'un programme.

2.4.5 Récapitulation

Le formalisme développé par GOODENOUGH et GERHART a donné lieu à de nombreuses variantes. Les notions de fiabilité et de validité qu'ils proposent rejoignent assez bien l'intuition, mais sont beaucoup trop fortes pour être applicables. Les variantes proposées dans la littérature tentent de rendre ces notions plus effectives. Dans le cas des tests révélateurs en réduisant le domaine considéré; dans le cas des tests discriminants, en tenant compte des programmes situés au voisinage du programme considéré.

2.5 Conclusion

Dans ce chapitre, nous avons tenté de formaliser la notion de test. Nous avons d'abord défini la correction d'un programme à partir de son comportement externe ("boîte noire"). Par contre, nous avons défini une erreur comme différence syntaxique entre le programme et un programme correct. A partir de cette notion, nous avons développé un modèle syntaxique d'erreur en distinguant trois classes : erreur de frontière, de calcul et de raffinement. Nous avons montré que ce formalisme est bien adapté au cas de programme "quasi-correct", typiquement ceux dont la correction peut être assurée en modifiant ou rajoutant une instruction par erreur.

Nous avons ensuite défini la notion de test, de fiabilité et de validité. Les notions sont cependant inapplicables dans la pratique. Aussi nous avons proposé deux variantes :

Tout d'abord, la notion de test révélateur d'un sous-domaine qui est bien adaptée pour évaluer les stratégies opérant par partionnement du domaine d'entrée, et plus spécialement les stratégies d'analyse de chemin (E1) ;

Puis la notion de test discriminant, issue des stratégies de test par mutation (D33), et bien adaptée au modèle d'erreur que nous avons développé.

Nous conclurons en soulignant que, de même que la notion d'erreur est étroitement dépendante du style de programme que l'on considère, les critères de qualité sont étroitement déterminés par la stratégie de validation que l'on envisage.

Bibliographie thématique : Validation de programmes par test

Nous présentons ici la bibliographie du rapport Validation de programmes par test : théorie et pratique [Boug 82].

Le lecteur trouvera des ouvrages concernant les thèmes suivant :

- Introduction au problème de la validation et de la certification ;
- Etude d'erreurs ; notion de fiabilité ;
- Notion de test ;
- Méthodologies de test de programmes ;
- Evaluation de la qualité ;
- Validation en matériel.

Cette bibliographie a été mise à jour en Janvier 1982.

A Introduction au problème de la validation et de la certification

(A1) W.C. HETZEL ;
A definitional framework.
in "Program Test Methods", ed. Hetzel, Prentice-Hall, 1973,
p. 7-10.

(A2) R.E. KEIRSTEAD, D.B. PARKER ;
On the feasibility of formal certification.
in "Program Test Methods", ed. Hetzel, Prentice-Hall, 1973,
p. 291-300.

(A3) S.L. GERHART ;
Program validation.
in "Computing System Reliability", ed. T. Anderson et B. Randell,
Cambridge, England : Cambridge, 1979,
p. 66-108.
*Complémentarité de la validation par test et de la preuve formelle.

(A4) E. GIRARD, G. MEMMI, J.C. RAULT ;
Génie logiciel, outils et techniques ; chapitre 4 : Techniques et outils du test et du contrôle de la qualité.
Publication de l'A.D.I.
*Introduction très complète et excellente synthèse des pratiques actuelles ; description des outils disponibles.

(A5) How computers fail.
I.E.E.E. Spectrum, vol. 18, n° 10, Octobre 1981, p. 44-46.
*Pour se faire peur .....

B Etude d'erreurs ; notions de fiabilité

(B1) J.D. GANNON, J.J. HORNING ;
Language design for programming reliability.
I.E.E.E., vol.SE-1, n° 2, juin 1975, p. 179-191.
* Expérimentation sur des langages-jouets.

(B2) J.D. GANNON ;
An experimental evaluation of data type convention
C.A.C.M., vol. 20, n° 8, août 1977, p. 584-595.
*Etude de l'influence (bénéfique) d'un langage fortement typé sur la programmation.

(B3) J.J. HORNING ;
Programming languages for reliable computing systems.
Working material for the international summer school on program construction, Munich, R.F.A., 26 juillet - 6 août 1978.
*Synthèse des études actuellement menées à partir de divers points de vue (tolérance à la faute, en particulier).

(B4) D.K. KOSY ;
Approaches to improved program validation through programming language design.
in "Program Test Methods", ed. Hetzel, Prentice-Hall, 1973, p. 75-92.

(B5) J. ARSAC ;
La construction de programmes structurés.
Dunod, Paris, 1977.

Etudes de fautes persistantes

(B11) R.L. GLASS ;
Persistent software errors.
I.E.E.E., vol. SE-7, n° 2, mars 1981, p. 162-168.

(B12) A. ENDRES ;
An analysis of errors and their causes in system programs.
I.E.E.E., vol. SE-1, n° 2, juin 1975, p. 140-149.
*Contient en particulier une classification des erreurs par types.

Notions de fiabilité

(B21) J.C. RAULT ;
An approach toward reliable software.
Proceedings of the 4th. International conference on software engineering, 17-19 Septembre 1979, Munich, R.F.A. p. 220-230.
*Synthèse de différentes approches, mettant en relief celles héritées du matériel.

(B22) J.C. RAULT ;
Le contrôle de la qualité et l'évaluation de la fiabilité des produits logiciels.
Publication de l'A.D.I.
*La fiabilité vue par les "industriels".

(B23) M.H. HALSTEAD ;
Elements of software science.
Elsevier, New-york, USA, 1977.
*Incidence de la complexité du programme sur le nombre d'erreurs.

(B24) S. HENRY, D. KAFURA ;
Software structure metrics based on information flow.
I.E.E.E., vol. SE-7, n° 5, septembre 1981, p. 510-518.

C Notion de test

(C1) J.B. GOODENOUGH, S.L. GERHART ;
Toward a theory of test data selection.
Siplan Notices, vol. 10, n° 6, juin 1975, p. 493-510.
*L'article de référence !

(C2) E.J. WEYUKER, T.J. OSTRAND ;
Theories of program testing and the application of revealing subdomains.
I.E.E.E., vol. SE-6, n° 3, mai 1980, p. 236-246.

(C3) E.J. WEYUKER ;
The oracle assumption of program testing.
Proceedings of the 13th Hawaii international conference on system Sciences, vol. 1, 1980, p. 44-49.

(C4) R. HAMLET ;
Reliability theory of program testing.
Acta Informatica, vol. 16, n° 1, 1981, p. 31-43.

(C5) T.J. OSTRAND, E.H. WEYUKER ;
Current directions in the theory of testing.
Proceedings of the COMPSAC 80, Chicago, Illinois, U.S.A., 27-31 Octobre 1980, p. 386-389.

D Methodologies de test de programmes

Méthodes de génération de données utilisant l'évaluation symbolique du programme.

(D1) W.E. HOWDEN ;
Methodology for the generation of program test data.
I.E.E.E., vol. C-24, n° 5, mai 1975, p. 554-559.

(D2) C.V. RAMAMOORTHY, S.F. HO, W.T. CHEN ;
Automated generation of program test data.
I.E.E.E., vol. SE-2, n° 4, décembre 1976, p. 293-300.

(D3) H.N. GABOW, S.N. MAHESHAVARI, L.J. OSTERWEIL ;
On two problems in the generation of program test data.
I.E.E.E., vol. SE-2, n° 3, septembre 1976, p. 227-231.
*Evaluation de la complexité des algorithmes de génération de chemins.

(D4) J. RICHARSON, L.A. CLARKE ;
A partition analysis to increase program reliability.
Proceedings of the 5th international conference on software Engineering, 9-12 Mars 1981, San Diego, California, U.S.A., p. 244-253.

(D5) D. GOOSENS ;
La méta-évaluation au service de la compréhension automatique de programmes.
Publication L.I.T.P. (Université Paris-7), n° 80-52, janvier 1981.

(D6) W.E. HOWDEN ;
Functional testing and design abstractions,
Journal of Systems and Software, n° 1, 1980, p. 307-313.

Systèmes automatiques utilisant cette stratégie

(D11) E.F. MILLER, R.A. MELTON ;
Automated generation of testcase datasets.
Sigplan Notices, vol. 10, N° 6, juin 1975, p. 51-58.

(D12) U. VOLGES, L. GMEINER, A. AMSCHLER ;
Sadat, an automated testing tool.
I.E.E.E., vol. SE-6, n° 3, mai 1980, p. 286-290.

(D13) L.A. CLARKE ;
A system to generate data and symbolically execute programs.
I.E.E.E., vol. SE-2, n° 3, septembre 1976, p. 215-222.

(D14) D. HEDLEY, M.A. HENNEL, M.R. WOODWARD ;
On the generation of test data for program verification.
in "Production and Assesment of Numerical Software", ed. M.A. Hennel et L.M. Delves, Academic Press, Londres, 1980.

(D15) W.E. HOWDEN ;
Dissect - a symbolic evaluation and program testing system.
I.E.E.E., vol. SE-4, n° 1, janvier 1978, p. 70-73.

Stratégie des domaines linéaires

(D21) L.J. WHITE, E.J. COHEN ;
A domain strategy for computer in program testing.
I.E.E.E., vol. SE-6, n° 3, mai 1980, p. 247-257.

(D22) S.H. ZEIL, L.J. WHITE ;
Sufficient test sets for path analysis testing strategy.
Proceedings of the 5th international conference on Software Engineering, 9-12 mars 1981, San Diego, California, U.S.A., p. 184-191.

Tests par mutation

(D31) T.A. BUDD, R.J. LIPTON, F.G. SAYWARD, R.A. DE MILLO ;
The design of a prototype mutation system for program testing.
AFIPS Conference Proceedings, vol. 47, 1978, p. 623-627.

(D32) R.A. DE MILLO ;
Mutation analysis as a tool for software quality assurance.
Proceedings of the COMPSAC 80, Chicago, Illinois, U.S.A.,
27-31 Octobre 1980, p. 390-393.

(D33) T.A. BUDD, R.A. DE MILLO, R.J. LIPTON, F.G. SAYWARD ;
Theorical and empirical studies on using program mutation to test the functional correctness of programs.
Proceedings of the 7th annual ACM symposium on principles of programming languages, Las Vegas, Nevada, U.S.A., 28-30 janvier 1980, p. 220-233.
* Article de référence contenant une bibliographie complète sur le sujet.

Tests guidés par les spécifications ; tests de compilateurs

(D41) J.A. BAUER, A.B. FINGER ;
Test plan generation using formal grammars.
Proceedings of the 4th international conference on software engineering, Munich, R.F.A., 17-19 septembre 1979, p. 425-432.

(D42) J.C. RAULT ;
La validation et la certification des compilateurs.
Publication de l'A.D.I.
* Article d'introduction générale, abordant les problèmes techniques, commerciaux et juridiques ; bibliographie.

(D43) B.A. WICHMANN, A.H.J. SALE ;
A Pascal processor validation suite.
Proceedings of the 2 - days workshop held at the National Computing Center, Manchester, Grande-Bretagne, 12-13 Septembre 1979.
* Description d'un jeu de test Pascal NPL.

(D44) J.B. GOODENOUGH ;
The ADA compiler validation capability.
Sigplan Notices, vol. 15, n° 11, 1980, p. 1-8.

(D45) B. HOUSSAIS ;
Un générateur de tests commandé par les grammaires ; manuel d'utilisation.
Publication IRISA, n° PI 119, juillet 1979, 46 pages.

(D46) B. HOUSSAIS ;
Un outil de production de tests pour compilateur.
Publication IRISA, TTI 57, 13 pages.

(D47) A.G. DUCAN, J.S. HUTCHINSON ;
Using attributed grammars to test designs and implementations.
Proceedings of the 5th international conference on software engineering, San Diego, California, U.S.A., 9-12 Mars 1981, p. 170-178.

(D48) A. CELENTANO, S. CRESPI REGHIZZI, P. DELLA VIGNA, C. GHEZZI, G. GRANATA, F. SAVORETTI ;
Compiler testing using a sentence generator.
Software - Practice and Experience, vol. 10, 1980, p. 897-918.

5D49) P. DESCHAMP ;
Perluette, a compiler producing system.
Proceedings of the 5th international symposium on programming, Torino, Italie, 6-8 avril 1982.

Tests par calculs formels

(D51) V. DONZEAU-GOUGE ;
Utilisation de la sémantique dénotationnelle pour la description d'interprétations non standard : application à la validation et à l'optimisation des programmes.
Transformation de programmes, 3ème colloque international sur la programmation, Paris, 28-30 mars 1978, ed. B. Robinet, Dunod, Paris.

(D52) J. HOREJS ;
Finite semantics : a technique for program testing.
Proceedings of the 4th international conference on Software Engineering, Munich, R.F.A., 17-19 Septembre 1979, p. 433-440.

(D53) R. CARTWRIGHT ;
Formal program testing .
Proceedings of the 8th annual symposium on the principles of programming languages, Williamsburg, Virginia, USA, 26-28 janvier 1981, p. 125-132.

E Evaluation de la qualité

Mesures structurelles

(E1) W.E. HOWDEN ;
Reliability of path analysis strategy.
I.E.E.E., vol. SE-2, n° 3, septembre 1979, p. 208-214.
*Evaluation selon les critères de GERHART et GOODENOUGH.

(E2) M.R. WOODWARD, D. HEDLEY, M.A. HENNEL ;
Experience with path analysis and testing of programs.
I.E.E.E., vol. SE-6, n° 3, Mai 1980, p. 278-285.
*Mesure de l'exhaustivité du test.

(E3) M. HENNEL ;
Quantitative measures of program testedness and complexity.
In "Production and assesment of numerical software", ed. M.A. Hennel et L.M. Delves, Academic-Press, 1980.

(E4) T.J. MAC CABE ;
A complexity measure.
I.E.E.E., vol. SE-2, n° 4, Decembre 1976, p. 308-320.
*Mesure fondée sur le graphe du programme.

Mesures statistiques

(E11) J.W. DURAN, S. NATFOS ;
A report on random testing.
Proceedings of the 5th international conference on software Engineering, San Diego, California, U.S.A., 9-12 mars 1981, p. 179-183.

(E12) J.W. DURAN, J.J. WIORKOWSKI ;
Quantifying software validity by sampling.
I.E.E.E., vol. R-29, n° 2, Juin 1980, p. 141-144.

(E13) M. PAIGE ;
Cost-effective software test methodologies.
Conference record of the 14th Asilomar,
Conference on circuits, systems and computers, Pacific Grove,
Califronia, U.S.A., 7-19 novembre 1980, p. 66-71.

(E14) A. VALLONE ;
Risk analysis of computer system desings.
Conference record of the 14th Asilomar, conference on the circuits,
systems and computers, Pacific Grove, California, U.S.A.,
17-19 Novembre 1980, p. 72-76.

(E15) A.L. GOEL, K. OKUMOTO ;
When to stop testing and start using software ?
Performance Evaluation Review, vol. 10, n° 1, 1981, p. 131-138.
*Modèle de développement et de test du logiciel.

Mesures issues des tests par mutation

(E21) W.E. HOWDEN ;
Completness criteria for testing elementary program functions.
Proceedings of the 5th international conference on software
Engineering, San Diego, California, U.S.A., 9-12 Mars 1981, p 235-243.

Evaluations expériementales de stratégies

(E31) W.E. HOWDEN ;
An evaluation of the effectiveness of symbolic testing.
Software - Practice and Expérience, vol. 8, 1978, p. 381-397.
*Comparaison de différentes méthodes de test utilisant
l'évaluation symbolique sur des programmes "réels".

F Validation en matériel

(F1) Manuels d'enseignement interne des sociétés productrices de matériel.
En particulier :
IBM manual 2521 "Fault detection", 30 Juin 1980.

(F2) P. GOEL ;
A implicit enumeration algorithm to generate tests for combinatorial circuits.
I.E.E.E., vol. C-30, n° 3, Mars 1981, p. 215-222.
*évaluation chiffrée de la puissance d'algorithmes de production de tests.

(F3) P. DUHAMEL, J.C. RAULT ;
Automatic test generation techniques for analog circuits and systems.
I.E.E.E., vol. CAS-26, n° 7, juillet 1979, p. 411-440.
*méthodes de test pour circuits analogiques ; bibliographie.

(F4) C.R. KIME ;
An abstract model for digital system fault diagnostic.
I.E.E.E., vol. C-28, n° 10, Octobre 1979, p. 754-767.

(F5) R.J. SPILLMAN ;
A continuous time model of multiple intermittent faults in digital systems.
Computer et electric engeneering, vol. 8, 1981, p. 27-40.

(F6) J.C. RAULT ;
Test et testabilité.
Publication de l'A.D.I.
*Synthèse de différentes méthodes de test en matériel ; comparaison de leur efficacité et des possibilités d'automatisation.

(F7) T.W. WILLIAMS, K.P. PARKER ;
Testing logic networks and designing for testability.
Computer, octobre 1979, p. 9-21.
*Excellente bibliographie.

(F8) C. DURANTE, N. GIAMBIASI, A. MIARA ;
Recherche d'une méthodologie générale de génération interactive de séquences de test de circuits logiques.
Publication du L.A.M., Montpellier.

(F9) Reliable systems : design and tests.
I.E.E.E., Spectrum, vol. 18 n° 10, octobre 1981, p. 50-72.

Application au logiciel

(F11) J.C. RAULT ;
Extension of hardware fault detection models to the verification of software.
In "Program test methods", ed. Hetzel, Prentice-Hall, 1973 , p. 255-262.

(F12) K.A. FOSTER ;
Error sensitive test cases analysis.
I.E.E.E., vol. SE-6, n° 3, mai 1980.

Introduction à la logique du premier ordre (égalitaire)

Cette annexe a pour but de préciser les principales définitions et propriétés de la logique du premier ordre (égalitaire) que nous utilisons dans ce travail.

Nous n'abordons, dans cette introduction, que les points strictement nécessaires à notre démarche. D'autre part, cette thèse ayant été motivée par des études concernant les types abstraits algébriques (cf. chapitres 7 et 8), nous préciserons, lorsque cela est possible, les liens étroits existant entre la logique du premier ordre et cette théorie.

Les définitions et théorèmes présentés sont, pour la plupart tirés de l'excellent livre de SHOENFIELD, Mathematical logic [Shoe 67], et du livre non moins intéressant de G. KREISEL et J.L. KRIVINE, Eléments de logique mathématique [K&K 66]. Nous conseillons très vivement au lecteur que cette rapide et superficielle introduction (ou plutôt recueil de définitions) laissera insatisfait, de se repor-

ter à ces ouvrages. Pour lui faciliter la tâche, de fréquentes références y seront faites au cours de l'exposé.

Pour le lecteur intéressé par une approche moins formelle, moins technique du problème, mais plus synthétique, voire philosophique, le livre de S.C. KLEENE, Mathematical logic [Klee 66] constitue un ouvrage d'excellente qualité, fourmillant de remarques historiques et épistémologiques extrêmement enrichissantes.

1. Langages

cf. [Shoe 67] paragraphe 2.4 p. 14 et ss.
cf [K&K 66] p. 14-15, 34

Un langage L du premier ordre (égalitaire) est la donnée d'une famille d'ensembles de symboles, tous distincts.

Cette famille contient tout d'abord un ensemble V de <u>variables</u>, notées $x, y, z, \ldots$, supposé infini.

Ensuite, pour chaque entier n, elle contient un ensemble F_n de symboles <u>fonctionnels</u> d'arité n, notés $f, g, \ldots$

Pour chaque entier n, elle contient un ensemble P_n de symboles de prédicat d'arité n, notés $p, q, \ldots$. L'on dit aussi symboles <u>relationnels</u>.

P_2 contient un symbole privilegié noté $=$ (d'où le nom de langage égalitaire).

Enfin, L contient un ensemble de symboles dits <u>logiques</u>, contenant le symbole $\neg$ d'arité 1, le symbole $\vee$ ("ou") d'arité 2, et pour tout élément x de V, le symbole "$\exists x$", d'arité 1.

Les symboles autres que les symboles logiques et les variables sont dits symboles non-logiques : $\bigsqcup_n F_n \cup P_n$

Un langage est dit un III-langage si l'ensemble de ses symboles non-logiques est

de cardinal $\aleph_1$ au plus ([Shoe 67], paragraphe 5.3).

Soit $L^1 = \langle V^1, (F_n^1)_{n\in\mathbb{N}}, (P_n^1)_{n\in\mathbb{N}} \rangle$ et $L^2 = \langle V^2, (F_n^2)_{n\in\mathbb{N}}, (P_n^2)_{n\in\mathbb{N}} \rangle$ deux langages du premier ordre. L^1 est dit une extension de L^2 si, pour tout n, $F_n^1 \supseteq F_n^2$ et $P_n^1 \supseteq P_n^2$, et $V^1 \supseteq V^2$. L^2 est dit alors une restriction de L^1. L'on note $L^2 \subseteq L^1$

Un langage peut donc être vu comme une signature Σ construite sur un seul type (d'où le nom de "premier ordre"). Néanmoins, la distinction entre symboles fonctionnels et relationnels (de prédicat) assure l'existence virtuelle d'un type Booléen partiellement prédéfini.

Un exemple de langage est le suivant.

$V = \{x, y, z, \dots\}$

$F_0 = \{0\}$, $F_1 = \{succ, pred\}$, $F_n = \emptyset$ $n \geq 2$

$P_0 = \emptyset$, $P_1 = \emptyset$, $P_2 = \{\geq, =\}$, $P_n = \emptyset$ $n \geq 3$

(vir connu)

2. Termes et formules

cf. [Shoe 67] paragraphe 2.4
cf [KEK 66] p. 14

Soit $L = \langle V, (F_n)_{n\in\mathbb{N}}, (P_n)_{n\in\mathbb{N}} \rangle$ un langage du premier ordre.

La notion de <u>terme</u> du langage est analogue à celle des types abstraits algébriques. [ADJ 78]

Plus précisément, soit A_1 l'alphabet suivant:

$A_1 = V \sqcup \bigsqcup_{n\in\mathbb{N}} F_n$

L'ensemble des termes de L, T_L, est le plus petit sous-ensemble de A_1^* vérifiant

- une variable est un terme,
- si $f \in F_n$, et $u_1, \dots, u_n$ sont des termes de L, alors $fu_1 \dots u_n$ est un terme de L.
  En particulier, si $f \in F_0$, f est un terme de L dit constant.

Une <u>formule atomique</u> peut être vue comme un terme de type Booléen. Plus précisément, soit $A_2 = V \sqcup \bigsqcup_{n\in\mathbb{N}} (F_n \sqcup P_n)$.

L'ensemble des formules atomiques de L, At_L, est le plus petit sous-ensemble de A_2^* tel que

- si $p \in P_n$ et $u_1 \dots u_n$ sont des termes de L alors $pu_1 \dots u_n$ est une formule atomique de L.

dans notre exemple,

succ (pred (x)) est un terme,

succ (y) aussi,

succ (pred (x)) $\geq$ succ (y) est une formule atomique.

De même,

succ (pred (x)) = x est une formule atomique.

Enfin, une <u>formule</u> est une expression complexe construite sur des formules atomiques. Plus précisément, soit $A_3 = V \sqcup \bigsqcup_{n \in \mathbb{N}} (F_n \cup P_n) \sqcup \{\neg, \vee\} \sqcup \bigsqcup_{x \in V} \{\exists x\}$

L'ensemble des formules de L, F_L, est le plus petit sous-ensemble de A_3^* tel que

- si ϕ est une formule atomique, ϕ est une formule,
- si ϕ et ψ sont des formules, $\phi \vee \psi$ en est une
- si ϕ est une formule, et x une variable, $\exists x\ \phi$ est une formule.
- si ϕ est une formule, $\neg\phi$ en est une.

Traditionnellement, on abrège :

$\neg \vee \neg \phi \neg \psi$ par $\phi \wedge \psi$ (lire "ϕ et ψ")

$\vee \neg \phi \psi$ par $\phi \rightarrow \psi$ (lire "ϕ implique ψ)

$\neg \exists x \neg \phi$ par $\forall x\ \phi$

$\wedge \vee \neg \phi \psi \vee \neg \psi \phi$ par $\phi \leftrightarrow \psi$ (lire "ϕ équivaut ψ")

Dans notre exemple

$\forall x \quad succ(x) \geq 0$ est une formule

$\neg(succ(x) \geq 0)$ en est une aussi

$\forall x \, [\, succ(x) \leq 0 \rightarrow x = 0]$ aussi.

Comme dans le formalisme des types abstraits, nous définissons la notion d'<u>occurence</u> d'une variable dans une formule.

Une occurence de la variable x est liée si elle est dans le champ d'un "$\exists x$" (qui peut être parfois caché dans un "$\forall x$"). Sinon elle est libre.

Une variable est dite liée dans une formule si toutes ses occurences sont liées.

Une formule est dite <u>fermée</u> si toutes ses variables sont liées.

Elle est dite <u>ouverte</u> si elle ne contient pas de quantificateur. (cf [Shoe 67] p. 19, 36).

$\forall x \forall y \, [\, succ(x) = succ(y) \rightarrow x = y]$ est fermée

$succ(x) \geq 0$ est ouverte

$succ(0) \geq 0$ est ouverte et fermée.

Si ϕ est une formule, x une variable et t un terme, $\phi_x[t]$ est la formule obtenue en substituant t à <u>toutes les occurences libres</u> de x dans ϕ.

5. Structures

cf. [Shoe 67] paragraphe 2.5
La notion de réalisation proposée en [KeK 66] pp. 14-15 est plus large, mais ne nous sera pas utile ici.

Une L-structure pour un langage L est l'analogue d'une Σ-algèbre pour une signature Σ

Soit $L = \langle V, (F_n)_{n \in \mathbb{N}}, (P_n)_{n \in \mathbb{N}} \rangle$ un langage du premier ordre, égalitaire.

Une structure $\mathcal{S}$ pour L est la donnée

- d'un ensemble de base, noté $|\mathcal{S}|$, dit "univers"; les éléments de $|\mathcal{S}|$ sont les individus;
- pour tout n, et pour tout $f \in F_n$, d'une application $f_{\mathcal{S}}$ de $|\mathcal{S}|^n$ dans $|\mathcal{S}|$; en particulier, si $e \in F_0$, $e_{\mathcal{S}}$ est un individu;
- pour tout n, et pour tout $p \in P_n$ sauf pour "=" $\in P_2$, un prédicat sur $|\mathcal{S}|^n$; on identifie ce prédicat avec le sous-ensemble de $|\mathcal{S}|^n$ sur lequel il est vrai.

De même que dans les types abstraits algébriques, l'on se restreint aux algèbres premières, de même il est souhaitable de pouvoir ici désigner dans le langage chaque individu.

Supposons L et $\mathcal{S}$ donnés, $\mathcal{S}$ étant une L-structure ; formons le langage $L(\mathcal{S})$ (ou $L(S)$ si $S = |\mathcal{S}|$) <u>canoniquement</u> à partir de L en rajoutant à L, comme symboles de constantes, les individus de $\mathcal{S}$ (notés $i, j, \ldots$). $\mathcal{S}$ s'étend alors canoniquement en une $L(\mathcal{S})$-structure en posant $i_{\mathcal{S}} = i$. $L(\mathcal{S})$ est une extension de L.

Lorsque nous parlerons de $L(\mathcal{S})$ (ou $L(S)$, avec $S = |\mathcal{S}|$) - structure, nous considérerons toujours une L-structure <u>étendue de cette manière</u>.

4. Valeur de vérité, sémantique

cf. [Shoe 67] p11, 19
cf [K&K 66] p. 15

Nous suivons ici l'approche décrite par [Shoe 67], suffisante pour notre propos. Celle proposée par [K&K 66] est plus complète et plus rigoureuse, mais beaucoup plus complexe. Pour retranscrire nos explications dans ce dernier cadre, il suffirait cependant de remplacer partout F par ϕ et T par $|\mathcal{S}|^V$

Dans le cadre des types abstraits, la sémantique est donnée par l'application, déduite de l'initialité, de T_Σ vers une Σ-algèbre donnée.

De même ici, soit L un langage, et $\mathcal{S}$ une L-structure.

A tout <u>terme</u> <u>sans variable</u> de L, $\mathcal{S}$ associe une valeur, un élément de $|\mathcal{S}|$.

Considérons maintenant ϕ <u>formule atomique</u> de L. Si ϕ est sans variable, et de la forme

- $u_1 = u_2$, où u_1 et u_2 sont des termes, nous posons $\mathcal{S}(\phi) = T$ si et seulement $\mathcal{S}(u_1) = \mathcal{S}(u_2)$;
- $p\, u_1 \ldots u_n$, où $u_1 \ldots u_n$ sont des termes, et $p \in P_n$, $p \neq$ "=",
  $\mathcal{S}(\phi) = T$ si et seulement si $p_{\mathcal{S}}[\mathcal{S}(u_1), \ldots, \mathcal{S}(u_n)]$

Considérons maintenant le cas d'une <u>formule</u> ϕ <u>fermée</u> (dont toutes les variables éventuelles sont liées).

Si ϕ est atomique, alors ϕ n'a pas de variable, et nous avons défini $\mathcal{S}(\phi)$.

Si ϕ est de la forme $\neg\psi$, $\mathcal{S}(\phi) = T$ si et seulement si $\mathcal{S}(\psi) = F$

Si ϕ est de la forme $\psi \vee \theta$, $\mathcal{S}(\phi) = T$ si et seulement si $\mathcal{S}(\psi) = T$ ou $\mathcal{S}(\theta) = T$, les deux l'étant éventuellement.

Enfin, si ϕ est de la forme $\exists x\, \psi$, $\mathcal{S}(\phi) = T$ si et seulement si

$\mathcal{S}[\psi_x[i]] = T$ pour un individu i de $L(\mathcal{S})$ au moins.

Il est alors facile de vérifier que $\wedge$, $\forall x$, $\rightarrow$, $\leftrightarrow$ on bien leur signification habituelle.

Dans notre exemple, considérons la structure $\mathcal{S}$ habituelle sur $\mathbb{Z}$. Alors

$\mathcal{S}(\forall x\, [succ(x) \geqslant 0]) = F$

$\mathcal{S}(\forall x \forall y\, [succ(x) = succ(y) \rightarrow x = y]) = T$

$\mathcal{S}(1+2) = 3$

$\mathcal{S}(0 \geqslant 1) = F$

5. Validité logique

Etant donné une signature Σ, et un système de Σ-axiomes E, le problème est de caractériser, d'étudier, les Σ-algèbres (premières) satisfaisant ces axiomes.

De même, en logique du premier ordre, étant donné un langage L et un ensemble de L-formules, l'on s'intéresse aux L-structures telles que ces L-formules y prennent la valeur T.

Plus précisément, étant donné un langage L et une structure $\mathcal{S}$, nous dirons qu'une $L(\mathcal{S})$-formule fermée ϕ est <u>valide</u> dans $\mathcal{S}$ si $\mathcal{S}(\phi) = T$.

Nous noterons $\mathcal{S} \models \phi$ ($\mathcal{S}$ valide ϕ).

Nous étendons cette définition au cas d'une formule quelconque $\phi(x_1 - x_n)$. Nous dirons que $\mathcal{S} \models \phi(x_1 - x_n)$ si

$\mathcal{S} \models \phi(i_1, \ldots, i_n)$ pour tout n-uplet d'individus de $\mathcal{S}$, $(i_1 - i_n)$.

Par exemple, $\mathcal{S} \models p(x) \wedge \exists x\, q(x)$ si, pour tout $s \in |\mathcal{S}|$, $\mathcal{S} \models p(s) \wedge \exists x\, q(x)$

(cf. [Shoe 67] p. 19).

Soit L un langage. Certaines L-formules ont la propriété d'être vraies dans toutes les L-structures. On les appelle les <u>axiomes logiques de L</u>. Ce sont (cf. [Shoe 67], p. 21)

① toutes les formules de la forme

$$\neg\phi \vee \phi$$

où ϕ est une formule

② toutes les formules de la forme

$$\phi_x[t] \rightarrow \exists x\, \phi$$

où ϕ est une formule, t un terme et x une variable. <u>Si x n'est pas libre dans ϕ</u>, il vient $\phi \rightarrow \exists x\, \phi$

③ toutes les formules de la forme

$$x = x$$

où x est une variable (identité)

④ toutes les formules de la forme

$$x_1 = y_1 \rightarrow \cdots \rightarrow x_n = y_n \rightarrow (f x_1 \cdots x_n = f y_1 \cdots y_n)$$

où $f \in F_n$, $x_1 \cdots x_n$, $y_1, \ldots, y_n$ étant des variables, et

$$x_1 = y_1 \rightarrow \cdots \rightarrow x_n = y_n \rightarrow (p x_1 \cdots x_n \rightarrow p y_1 \cdots y_n)$$

où $p \in P_n$, $x_1 \cdots x_n$ $y_1 \cdots y_n$ étant des variables (égalité)

Il y en a donc une (grande!) infinité.

6. Deduction formelle

cf. [Shoe 67] p. 21, 22, 23

Si l'on considère un langage et une structure $\mathcal{S}$ de ce langage, et une famille Γ de formules valides sur $\mathcal{S}$, l'on peut déduire formellement de Γ d'autres formules valides sur $\mathcal{S}$ (théorème de validité).

L'adjectif "formel" signifie que, dans cette opération, l'on ne tient compte que de l'aspect syntaxique des formules, et non de la structure $\mathcal{S}$ considérée. Ce processus est donc analogue aux méthodes de réécriture de termes dans les types abstraits, ou plus généralement, dans un système de réécriture.

De très nombreuses règles de déduction peuvent être citées (cf. [Klee 66], théorèmes 2 et 3 pp. 15, 18).

Nous en considérerons 5. Soit φ, ψ, θ des formules.

① De φ, on peut déduire $\psi \vee \varphi$.

② De $\varphi \vee \varphi$, on peut déduire φ.

③ De $\varphi \vee (\psi \vee \theta)$, on peut déduire $(\varphi \vee \psi) \vee \theta$

④ De $\neg\varphi \vee \theta$ et $\varphi \vee \psi$, on peut déduire $\psi \vee \theta$

⑤ Si x n'est pas libre dans ψ,
de $(\exists x\ \varphi) \to \psi$,
on peut déduire $\varphi \to \psi$

Nous employons le symbole $\vdash$ pour désigner une déduction formelle : $\varphi \vdash \psi \vee \varphi$.

Nous étendons la signification de $\vdash$ à une déduction complexe. Soit Γ un ensemble de formules, et φ une formule.

φ est démontrable à partir de Γ, si il existe une suite de formule $\varphi_0, \dots, \varphi_k$ telle que pour tout i, $0 \leq i \leq k$,

$\varphi_i \in \Gamma$ ou

φ_i est déductible de $\{\varphi_0, \dots, \varphi_{i-1}\}$ par application de l'une des cinq règles précédentes :
$\{\varphi_0, \dots, \varphi_{i-1}\} \vdash \varphi_i$

Par abus de langage, nous noterons aussi $\vdash$ une déduction complexe.

Montrons par exemple la commutativité de $\vee$.
Soit Γ l'ensemble des axiomes logiques de L, et A, B des L-formules.

$\Gamma, A \vee B \vdash \neg A \vee A$ (axiome ① pour A)

$\Gamma, A \vee B \vdash A \vee B$.

$\Gamma, A \vee B \vdash B \vee A$ (par la règle ④ avec $\Theta = A$)

Examinons quelques propriétés de $\vdash$

Soit Γ et A, Γ étant un ensemble de formules et A une formule.

Si $\Gamma \vdash A$, alors $\Gamma' \vdash A$ pour tout $\Gamma' \supseteq \Gamma$

Une preuve (déduction formelle complexe) n'utilisant qu'un nombre fini de formule, <u>si A est démontrable à partir de Γ, A l'est à partir d'un sous-ensemble fini de Γ</u>. C'est une forme du <u>théorème de compacité</u> de la logique du premier ordre égalitaire, qui tient une place centrale dans notre travail (cf 3.2.4., 4.4.3. etc.). Nous y reviendrons.

Soit $\Gamma_1, \Gamma_2, \Gamma_3$ trois ensembles de L-formules. Etendons la signification du symbole $\vdash$ de la manière suivante.

$\Gamma_1 \vdash \Gamma_2$ si <u>toute</u> formule de Γ_2 est démontrable à partir de Γ_1.

Alors, si $\Gamma_1 \vdash \Gamma_2$, et $\Gamma_2 \vdash \Gamma_3$,

$\Gamma_1 \vdash \Gamma_3$.

La relation $\vdash$ entre les ensembles de formules est donc <u>transitive</u>.

7. Théories

cf. [Shoe 67] p. 22.

L'un des objets de la logique du premier ordre (égalitaire) est l'étude des relation entre propriétés logiques ($\models$) et propriétés formelles ($\vdash$).

Dans l'étude de ce problème, les axiomes logiques jouent un rôle particulier, puisqu'ils sont valides dans toute structure.

L'on se restreint donc habituellement à des ensembles de L-formules contenant au moins les L-axiomes logiques. De tels ensembles sont appelés des L-théories du premier ordre

Plus généralement, une théorie T égalitaire ~~du~~ premier ordre est la donnée

- d'un langage égalitaire du premier ordre $L(T)$,
- d'un ensemble de formules (d'axiomes) contenant au moins tous les axiomes logiques de $L(T)$,
- des cinq règles de déduction formelle exposée au paragraphe 6;

Les axiomes autres que logiques sont dits <u>non-logiques</u>.

Une fois le langage L fixé, une théorie est définie par l'ensemble de ses axiomes non-logiques. Nous identifierons en général l'un et l'autre. En particulier, $T = \emptyset$ désigne la L-théorie n'ayant comme axiomes que les L-axiomes logiques.

Un modèle de T est une structure pour L(T) telles que <u>tous les axiomes de T</u> soient valides. Il suffit bien évidement que les axiomes non-logiques de T le soient.

Si T_1 et T_2 sont des L-théories, nous notons $T_1 \cup T_2$ la L-théorie ayant pour ensemble d'axiomes non-logiques la réunion de celui de T_1 et de celui de T_2.

Soit L un langage, T une L-théorie, A une L-formule.

<u>A est un théorème de T</u> si $T \vdash A$.
A est dite conséquence formelle de T.

<u>A est valide dans T</u> si, pour tout modèle $\mathcal{S}$ de T, $\mathcal{S} \models A$. A est dite conséquence logique de T; on notera $T \models A$.

Le théorème de validité ([Shoe 67] p.23]) implique que si A est un théorème de T,

A est valide dans T. Les propriétés formelles sont donc, en général, plus contraignantes que les propriétés logiques correspondantes.

Examinons le comportement de $\vdash$ vis à vis de $\sqcup$. Il est conforme à l'intuition.

Soit T_1, T_2, T_3 trois théories sur un langage donné.

$T_1 \vdash T_2 \sqcup T_3$ si et seulement si

$T_1 \vdash T_2$ et $T_1 \vdash T_3$

Si $T_1 \vdash T_3$, alors $T_1 \sqcup T_2 \vdash T_3$;

la réciproque étant bien sûr fausse en général.

Des propriétés analogues sont vérifiées avec $\models$ au lieu de $\vdash$

Soit T_1 et T_2 deux L-théories.

T_1 et T_2 sont dites <u>logiquement équivalentes</u> si elles ont les mêmes modèles : $\mathcal{S} \models T_1$ si et seulement si $\mathcal{S} \models T_2$. Nous noterons $T_1 \vDash T_2$.

T_1 et T_2 sont dites <u>formellement équivalentes</u> si $T_1 \vdash T_2$ et $T_2 \vdash T_1$. L'on note ceci $T_1 \dashv\vdash T_2$.

Soit T une troisième L-théorie.

T_1 et T_2 sont dites <u>T-logiquement équivalentes</u> si, parmi les modèles de T, elles sont les mêmes modèles. C'est équivalent à $T \sqcup T_1 \vDash T \sqcup T_2$

De même, nous parlerons de T-formelle équivalence si $T \sqcup T_1 \vdash T \sqcup T_2$.

Les propriétés formelles entraînent les propriétés logiques correspondantes.

Soit T et T' deux L-théories. $T \vdash T'$ si et seulement si <u>tous</u> les axiomes non-logiques de T' sont démontrables à partir de T.

En particulier, si T ne contient qu'un nombre fini d'axiomes non-logiques, $A_1, \ldots, A_n$, T est formellement équivalente à T', L(T)-théorie ne contenant qu'un seul axiome non-logique, $A_1 \wedge \ldots \wedge A_n$.

8. Les grands théorèmes

Nous présentons ici les grands théorèmes que nous utilisons, parfois implicitement, dans notre travail, ou qui justifient certains de nos choix.

Nous ne donnons bien sûr que l'essentiel, le lecteur intéressé, passionné, "tatillon" ou incrédule pouvant tout à loisir se reporter aux références citées.

8.1. Théorème de validité

Théorème ([Shoe 67] p.23)

Soit L un langage (égalitaire du premier ordre), T une L-théorie.

Tout théorème de T est valide dans T

Autrement dit, si A est une L-formule démontrable à partir de T, A est valide dans toute structure modèle de T.

8.2. Théorème de déduction ([Shoe 67], paragraphe 3.3)

Théorème

Soit L un langage (égalitaire du premier ordre),

Soit T une L-théorie, $A_1, \dots, A_k, B$ des L-formules.

$T \vdash A_1 \to \dots \to A_k \to B$ si et seulement si $T[A_1, \dots, A_k] \vdash B$

$T[A_1, \dots, A_k]$ désigne la L-théorie dont l'ensemble des axiomes non-logiques est la réunion de celui de T et de $\{A_1, \dots, A_k\}$.

Ce théorème signifie que pour démontrer $A_1 \to \dots \to A_k \to B$ à partir de T, l'on peut supposer $A_1, \dots, A_k$ et démontrer B.

8.3. Théorème de réduction prénexe

([Shoe 67] paragraphe 3.5
[K&K 66] p. 17)

Théorème :

Soit L un langage (égalitaire du premier ordre), et soit A une L-formule.

Alors il existe une L-formule ouverte $B(x_1 - x_n)$ telle que, sous les axiomes logiques de L, A soit formellement equivalent à une formule de la forme $Q_1 x_1 \dots Q_n x_n \, B(x_1 - x_n)$, où les Q_i sont des quantificateurs existentiels ou universels.

L'on peut de plus choisir B de la forme

$$C_1 \wedge \dots \wedge C_k ,$$

et chaque C_i de la forme

$$D_1 \vee \dots \vee D_{p_i} \vee \neg E_1 \vee \dots \vee \neg E_{q_i} ,$$

les D_ℓ et E_ℓ étant des L-formules atomiques. (forme normale conjonctive).

Si tous les Q_i sont universels, A est dite <u>purement universelle</u>; s'ils sont tous existentiels, A est dite <u>purement existentielle</u>.

Une telle forme peut se noter en abrégé :

$$\bar{Q}\,\bar{x} \wedge (\vee \bar{\alpha} \vee \neg \bar{\beta}) ;$$

en remarquant que $\vee\bar{\alpha} \vee \neg\bar{\beta}$ s'écrit

$$\wedge\bar{\beta} \rightarrow \vee\bar{\alpha} ,$$ il vient

$$\bar{Q}\,\bar{x} \wedge (\wedge\bar{\beta} \rightarrow \vee\bar{\alpha}),$$

ce qui peut s'interpréter comme une structure "CASE" (en Pascal, par exemple) sur le vecteur $\bar{x}$.

3.4. Théorème de complétude

([Shoe 67] paragraphe 4.2)

<u>théorème</u> :

Soit L un langage (égalitaire du premier ordre)
Soit A un L-formule, et T une L-théorie.
A est un théorème de T <u>si et seulement si</u> A est valide pour T

Autrement dit, A est démontrable à partir des axiomes de T et des 5 règles d'inférence si et seulement si A est valide dans toutes les structures modèles de T. C'est donc une réciproque au théorème de validité.

La démonstration de ce théorème est longue, difficile, et extrêmement élégante dans la version proposée par SHOENFIELD (et due à HENKIN). Elle fait, de toute façon, intervenir de manière cruciale l'axiome du choix, par l'intermédiaire du théorème de ZORN ici, si l'on veut traiter le cas général (non dénombrable).

Une application utile de ce théorème sera la suivante (cf. 5.1.2)

<u>proposition</u> (inspirée de [Shoe 67] p.33 et p.55)

Soit L_0 un langage (égalitaire du premier ordre), L une extension de L_0.
Soit T une L-théorie dont tous les axiomes non-logiques sont des L_0-formules.
Soit A une L_0-formule.
Supposons $T \vdash A$
Alors $T|L_0 \vdash A$.

Précisons que $T|L_0$ est la <u>L_0-théorie</u> ayant les mêmes axiomes non-logiques que T

<u>démonstration</u> :

Par le théorème de complétude, il suffit de montrer que toute L_0-structure $\mathcal{S}$ modèle de $T|L_0$ valide A.

Soit $\mathcal{S}_0$ une telle L_0-structure. Donnant des significations quelconques aux symboles de $L \setminus L_0$, étendons $\mathcal{S}_0$ en une L-structure $\mathcal{S}$ (sur le même ensemble de base). Donc $\mathcal{S}$ valide tous les axiomes non logique de T, car ce sont ceux de $T|L_0$; $\mathcal{S}$ valide les axiomes logiques de T par définition. Donc $\mathcal{S} \models T$. Puisque $T \vdash A$, par le théorème de validité, $\mathcal{S} \models A$. Mais A étant une L_0-formule, cela signifie que $\mathcal{S}_0 \models A$!

▨

8.5. Théorème de non-catégoricité

([Shoe 67], paragraphe 5.3)

Comme le laisse penser l'efficacité du théorème de complétude, une théorie a de très nombreux et très divers modèles.

Théorème

Soit L un $\mathfrak{m}$-langage (égalitaire du premier ordre), $\mathfrak{m}$ étant un cardinal infini (cf. la première partie de cette annexe).
Soit T une L-théorie.
Si T admet un modèle, T admet un modèle de cardinal $\mathfrak{n}$, pour tout cardinal $\mathfrak{n} \geqslant \mathfrak{m}$

En particulier, $\mathbb{N}$ n'est pas axiomatisable dans la logique du premier ordre (mais l'est au second ordre, cf. axiomes de PEANO).
Pour prendre le vocabulaire des types abstraits algébriques, $\mathbb{N}$ n'est pas axiomatisable sans l'introduction d'un type et de fonctions cachés, en l'occurence, le type "Ensemble d'entiers".

Dans le cadre de notre travail, une conséquence de ce théorème est la suivante.

Soit $\mathcal{C} = \langle L, S, (\mathcal{S}), A \rangle$ un contexte de test, et soit $\mathcal{T} = \langle H, (T_n)_{n \in \mathbb{N}} \rangle$ un jeu de tests sur $\mathcal{C}$. Considérons par exemple la validité asymptotique.

Le fait que $\mathcal{T}$ soit logiquement asymptotiquement valide (cf. 3.2.) s'exprime par la propriété suivante.

Pour toute structure $\mathcal{S}$ de $(\mathcal{S})$,

si $\mathcal{S} \models \bigcup_{n \in \mathbb{N}} T_n$, alors $\mathcal{S} \models A$.

Supposons, un instant, que $(\mathcal{S})$ contienne tous les modèles de $H \bigcup_{n \in \mathbb{N}} T_n$. Puisque toute structure de $(\mathcal{S})$ valide H, de la propriété précédente et du théorème de complétude, l'on peut déduire $H \bigcup_{n \in \mathbb{N}} T_n \vdash A$.

Dans ce cas, la validité asymptotique logique entraînerait la validité asymptotique formelle!

Mais toute les structures de $(\mathcal{S})$ étant construites sur le même ensemble de base S (infini) et $H \bigcup_{n \in \mathbb{N}} T_n$, par le théorème précédent, admettant au moins un modèle de cardinal strictement supérieur à celui de S s'il admet un modèle, notre supposition n'est <u>jamais</u> vraie dès qu'il existe un $\mathcal{S}$ telle que $\mathcal{S} \models \bigcup_{n \in \mathbb{N}} T_n$ (car alors $H \bigcup_{n \in \mathbb{N}} T_n$ a au moins $\mathcal{S}$ comme modèle).

Quelle que soit la "catégoricité" des hypothèses d'un jeu de tests, les propriétés formelles sont donc toujours strictement plus contraignantes en général que leur duale logique.

8.6. Théorème de compacité

([Shoe 67], paragraphe 5.1,
[K&K 66], pp. 21, 22, 35)

Théorème :

Soit L un langage (égalitaire du premier ordre), T une L-théorie, A une L-formule.
Si A est un théorème de T, A est un théorème d'une sous-théorie T' finie de T.
Si A est valide dans T, A est valide dans une sous-théorie T' finie de T

La première partie du théorème exprime simplement que la preuve de A dans T ne fait intervenir qu'un nombre fini d'axiomes (une théorie finie est une théorie ne contenant qu'une quantité finie d'axiomes non-logiques).
La seconde partie découle du théorème

de complétude. Une démonstration directe est cependant possible. Difficile mais très élégante, elle fait intervenir le théorème de Tychonov sur les ultrafiltres dans les compacts (d'où le nom du théorème), et donc, par ce biais, l'axiome du choix (cf. [Shoe 67], exercice 30 p. 105).

Une application importante de ce théorème est la suivante.

<u>proposition</u>

Soit L un langage (égalitaire du premier ordre), S un ensemble.

Supposons que L contienne un symbole de prédicat unaire p.

Soit T_1 la $L(S)$-théorie ayant pour seul axiome non-logique $\forall x\, p(x)$.

Soit T_2 la $L(S)$-théorie ayant pour axiomes non-logiques les formules $p(s)$, avec s parcourant S.

T_1 et T_2 <u>ne sont pas</u> formellement équivalentes.

Et pourtant, pour toute $L(S)$-structure $\mathcal{S}$, $\mathcal{S} \models T_1$ si et seulement si $\mathcal{S} \models T_2$!

<u>démonstration</u>.

Par le théorème de complétude, si elles étaient formellement équivalentes, elles auraient les mêmes modèles.

Soit a un élément n'appartenant pas à S. Considérons $S \sqcup \{a\}$, et prenons pour $\mathcal{S}$ la structure suivante :

son ensemble de base est $S \sqcup \{a\}$

$p(x)a$ pour interprétation $\neg x = a$.

Alors $\mathcal{S} \models T_2$, mais $\mathcal{S} \not\models T_1$

(une démonstration plus élémentaire est possible)

$\blacksquare$

En particulier, si $S = \mathbb{N}$,

$\{\forall x\, p(x)\}$ <u>n'est pas démontrable</u> à partir de $\{p(0), p(1), \ldots\}$

9. Conclusion

Nous avons, dans cette annexe, esquissé à gros traits les principales notions de logique (égalitaire) du premier ordre dont nous avons besoin au cours de ce travail.

La logique du premier ordre nous fournit un outil mathématique permettant de développer un formalisme original, rigoureux et fondé, autour de la notion de test. Cet outil est déjà bien connu, bien dominé et largement utilisé. nous avons à notre disposition de nombreux théorèmes et contre-exemples, de nombreuses et puissantes techniques de démonstration bien délimitées, un jeu de définitions parmi lesquelles nous pouvons puiser selon nos besoins.

Le fait de pouvoir disposer d'une telle base mathématique nous paraît constituer un atout important pour la réalisation, la diffusion, l'impact d'un tel travail de formalisation.

Nous sommes cependant conscient que l'utilisation d'une base mathématique déjà structurée rigidement conditionne, pour une large part le style, la forme de nos résultats.

Ainsi l'outil détermine-t-il pour une large part les rapports entre sujet et objet. Rappelons brièvement les principales conséquences de cette influence (cf. 2.2.).

Pour respecter l'esprit de ce formalisme, nous sommes amené à considérer des théories plutôt que de simples ensembles d'axiomes, ce qui alourdit la présentation, d'autant plus que l'usage que nous faisons de cette notion ne concerne pratiquement que les parties non-logiques des théories.

Ensuite, la dualité entre propriétés logiques et formelles nous conduit à des énoncés lourds et peu élégants. Cependant, elle nous permet de décrire une stratégie de construction précise (cf. 5.4.).

Enfin, cette approche du problème privilégie les applications issues du formalisme des types abstraits algébriques, proche de celui de la logique du premier ordre. Le choix d'une autre base mathématique eût sans doute conduit à privilégier d'autres formalismes.

[ADJ 78] J.A. GOGUEN, J.W. TATCHER, E.G. WAGNER;
An initial algebra approach to the specification, correctness, and implementation of abstract data types;
in Current trends in programming, methodology, vol. 4, ed. R.T. YEH,
Prentice-Hall, New-Jersey, 1978.

[Poido 81] M. POIDOIT;
Une méthode de présentation des types abstraits : applications;
Thèse de troisième cycle, Université Paris-Sud, 1981.

[Boug 82] L. BOUGE;
Validation de programmes par test : théorie et pratique;
Rapport LITP, Universités Paris 6 et 7,
no 82-18, Avril 1982.

[Budd 81] T.A. BUDD ;
Mutation analysis : ideas, examples, problems and prospects ;
in Computer program testing, ed. B. CHANDRASEKARAN, S. RADICCHI,
North-Holland, 1981, p. 129-148

[CIP 80] H. WIRSING, M. BROY ;
Abstract data types as lattices of finitely generated modules ;
Proceeding of Mathematical Foundations of Computer Sciences, Rydzyna Septembre 1980.

[Culi 80] K. CULIK II ;
Homomorphisms : Decidability, Equality and Test Sets ;
Rapport de l'université de Waterloo, Canada, CS-80-02, janvier 1980.

[Demo 82] R. DEMOLOMBE ;
Utilisation du calcul des prédicats comme langage d'interrogation des bases de données ; contribution à l'analyse et à l'évaluation des questions ;
Thèse d'état, Université Paul Sabatier, Février 1982, Toulouse.

[Desc 80] Ph. DESCHAMP ;
Production de compilateurs à partir d'une description sémantique des langages de programmation : le système Perluette ;
Thèse de docteur-ingénieur, Institut National Polytechnique de Lorraine, Nancy, Octobre 1980.

[Gaud 79] M. C. GAUDEL ;

Algebraic specification of abstract data types;
Rapport INRIA, no 360, Août 1979.

[Gaud 80] M.C. GAUDEL;
Génération et preuve de compilateurs basées sur une sémantique formelle des langages de programmation;
Thèse d'Etat, Institut National Polytechnique de Lorraine, Nancy, Mars 1980.

[Gell 78] M. GELLER;
Test data as an aid in proving program correctness;
CACM, vol 21, no 5, mai 1978.

[GMH 81] J. GANNON, P. McMULLIN, R. HAMLET;
Data-abstraction implementation, specification and testing;
ACM TOPLAS, vol 3, no 3, juillet 1980, p. 211-223.

[GMHA 80] J. GANNON, P. McMULLIN, R. HAMLET, M. ARDIS;
Testing Traversable Stacks;
Sigplan Notices, vol 15, no 1, janvier 1980, p. 58-65.

[Gogu 77] J.A. GOGUEN ;
Abstract errors for abstract data types ;
in Formal description of programming concepts, ed. NEUHOLD,
North-Holland, 1978, p. 491-526.

[Gutt 80] J.V. GUTTAG ;
Notes on type abstraction (version 2) ;
IEEE, vol. SE-6, no. 1, 1980

[G&G 75] J.B. GOODENOUGH, S.L. GERHART;
Toward a theory of test data selection;
Sigplan Notices, vol 10, no 6, juin 1975,
p. 493-510.

[G&H 78] J.V. GUTTAG, J.J. HORNING;
the algebraic specification of
abstract data types;
Acta informatica, vol. 10, 1978

[Klee 66] S.C. KLEENE;
Mathematical logic;
John Wiley & Sons, USA, 1966.

[K&K 66] G. KREISEL, J.L. KRIVINE;
Eléments de logique mathématique;
théorie des modèles;
Dunod, Paris, 1966.

[Mann 74] Z. MANNA;
Mathematical theory of computation;
Mc Graw-Hill, USA, 1974.

[Nico 79] J.M. NICOLAS;
Contributions à l'étude théorique des bases de données : apports de la logique mathématique;
Thèse d'état, Université Paul Sabatier, Toulouse, Decembre 1979.

[Revu 79] D. REVUZ;
Tests statistiques;
Polycopié de cours, Université Paris-7, 1979.

[Reyn 81] J.C. REYNOLDS;
The craft of programming;
Prentice-Hall, Londres, 1981

[Weyu 80] E.J. WEYUKER;
The oracle assumption of program testing;
Proceedings of the 13th Hawaii international conference on system Sciences, vol 1, 1980, p. 44-49.

[Whit 81] L.J. WHITE;
Basic mathematical definitions and results in testing;
in Computer program testing,
ed. B. CHANDRASEKARAN, S. RADICCHI
North-Holland, 1981.

[W&O 80] E.J. WEYUKER, T.J. OSTRAND;
Theories of program testing and the application of revealing sub-domains;
IEEE, vol. SE-6, no 3, mai 1980,
p. 236-246.

[Schoe 67] J.R. Schoenfield

Outil de validation automatique de spécifications : listages et exemples

Au chapitre 8, nous avons décrit le principe et l'implantation d'un outil automatique de validation de spécifications, fondé sur la modélisation présentée dans cette thèse.

Nous présentons dans cette annexe les listages de quelques points importants de cette implantation :

- présentation et algèbre sous test, dans le cas proposé au chapitre 7,
- strategie d'instanciation des candidats
- fonction d'évaluation des candidats totalement instanciés.

De plus, nous présentons un listage illustrant (partiellement, faute de place) le comportement de cet outil lors du test des trois axiomes du type File sur l'algèbre considérée.

Présentation et algèbre sous test (cf. 8.4.1, 8.4.8)

```
;exemple tire du rapport 360 INRIA de M.C.Gaudel
;
;type File

;ORDRE definit la hierarchie de la presentation

(setq ORDRE '(File Entier))


;TYPES definit la presentation

(setq TYPES
      '((Entier
          (Operations
            (Croissantes ())
            (Constantes ((/0 () Entier)))
            (Decroissantes ())
            (Metas ((meta_Entier_+ () Entier)
                    (meta_Entier_- () Entier)))
            (Etales ((pred (Entier) Entier)
                     (succ (Entier) Entier))))
          (Axiomes (Variables ((i Entier)))
                   (Formes ((1 (EGAL (pred (succ i)) i))
                            (2 (EGAL (succ (pred i)) i))))))
        (File
          (Operations
            (Croissantes ())
            (Metas ((meta_File_1 () File)))
            (Constantes ((file_vide () File)))
            (Decroissantes ((premier (File) Entier (Preconditions))))
            (Etales ((ajouter (File Entier) File)
                     (enlever (File) File))))
          (Axiomes
            (Variables ((q File) (r File) (i Entier)))
            (Formes
              ((1 (EGAL (enlever (ajouter q i))
                        (SAS (EGAL q (file_vide))
                             (file_vide)
                             (ajouter (enlever q) i))))
               (2 (EGAL (enlever (file_vide)) (file_vide)))
               (3 (EGAL (premier (ajouter q i))
                        (SAS (EGAL q (file_vide)) i (premier q)))
                  (Preconditions (premier))))))
          (Preconditions
            (Variables ((q File)))
            (Formes ((premier (q) (null (EGAL q (file_vide))))))))))))))

;et voici pour la presentation!
```

```
;Passons maintenant a la definition de l'algebre sous test

;Tout d'abord les structures de base

(defun SAS (b x y)
       (cond ((eq b t) x)
             ((eq b nil) y)
             (t (error "Mauvais arguments pour SAS: " (list b x y)))))))

(defun EGAL (x y) (equal x y))))))


;Les fonctions de representation pour File(Entier)

;Le support du type Entier est l'ensemble des objets Lisp de type fixnum.

(defun est_un_Entier (i)
       (eq (typep i) 'fixnum))))

(defun /0 () 0)

(defun succ (i)
       (cond ((est_un_Entier i) (1+ i))
             (t (error "Mauvais argument pour succ: " i)))))))))

(defun pred (i)
       (cond ((est_un_Entier i) (1- i))
             (t (error "Mauvais argument pour pred: " i))))))))))

(defun meta_Entier_+ () (random  10))))

(defun meta_Entier_- () ( - 0 (random 10)))))



;Le support du type File est l'ensemble des listes de fixnums

(defun est_une_File (q)
       (or (eq q ())
           (and (listp q)
                (est_un_Entier (car q))
                (est_une_File (cdr q))))))

(defun file_vide () ())

(defun premier(q)
       (cond ((and (est_une_File q)
                   (null (eq q ())))
              (car q))
             (t (error "Mauvais argument pour premier: " q))))))))

(defun enlever (q)
       (cond  ((est_une_File q) (cdr q))
              (t (error "Mauvais argument pour enlever: " q))))))

(defun ajouter (q i)
       (cond ((and (est_une_File q) (est_un_Entier i)) (append q (list i)))
             (t (error "Mauvais arguments pour ajouter " (list q i)))))))

(defun meta_File_1 ()
       '(1 -4 6)))))


;ET VOILA!
```

Stratégie d'instanciation (cf. 8.4.7.)

```
;Cette fonction implante le coeur de la strategie. Elle rend la liste des
;instanciations du candidat selon la vartypee, qui a ete prealablement
;enlevee de la liste.

(defun strategie_1_candidat_1
       (candidat vartypee nom_type)
       (cond ((eq (distinction vartypee) 1)        ;Il faut redescendre au type
                                                   ;inferieur
              (append
                (instancier_candidat_Croissantes
                  candidat
                  vartypee
                  nom_type)
                (instancier_candidat_Metas
                  candidat
                  vartypee
                  nom_type)
                (instancier_candidat_Constantes
                  candidat
                  vartypee
                  nom_type)))
             ((eq (distinction vartypee) ())       ;C'est une variable d'un
                                                   ;type inferieur.
              (append
                (instancier_candidat_Constantes
                  candidat
                  vartypee
                  nom_type)
                (instancier_candidat_Metas
                  candidat
                  vartypee
                  nom_type)
                (instancier_candidat_Croissantes
                  candidat
                  vartypee
                  nom_type)))
             ((and (eq (typep (distinction vartypee)) 'fixnum)
                   (greaterp (distinction vartypee) 1))
              (append
                (instancier_candidat_Etales
                  candidat
                  vartypee
                  nom_type
                  (1- (distinction vartypee)))
                (instancier_candidat_Croissantes
                  candidat
                  vartypee
                  nom_type)
                (instancier_candidat_Metas
                  candidat
                  vartypee
                  nom_type)
                (instancier_candidat_Constantes
                  candidat
                  vartypee
                  nom_type)))
             (t (error "Mauvaise distinction: " vartypee)))))))))))))))
;OUF!
```

Fonction d'évaluation (cf 8.4.6.)

```
(defun evaluer_1 (candidat nom_type)
       (let ((Exigences (assoclist_car 'Exigences candidat))
             (liste_Contraintes (cadr (assoclist_car
                                        'Contraintes
                                        (cadr Exigences))))
             (liste_Echecs (cadr (assoclist_car
                                   'Echecs
                                   (cadr Exigences))))
             (liste_Preconditions (cadr (assoclist_car
                                          'Preconditions
                                          (cadr Exigences))))
             (liste_listes_formes (cadr (assoclist_car 'Termes candidat))))
            (cond ((null (evaluer_liste_atome
                           liste_Contraintes t))
                   (terpri)
                   (princ "Contraintes non valides! "))
                  ((null (evaluer_liste_atome
                           liste_Preconditions t))
                   (terpri)
                   (princ "Preconditions non valides! "))
                  ((null (evaluer_liste_atome
                           liste_Echecs nil))
                   (cond ((null (evaluer_liste_listes_erreur
                                  liste_listes_formes))
                          (terpri)
                          (princ "Cas d'echec valide, mais evaluation")
                          (terpri)
                          (princ "menee a terme! ")
                          (experience_echouee))
                         (t (terpri)
                            (princ "Cas d'echec valide, mais evaluation")
                            (terpri)
                            (princ "avortee: experience reussie! "))))
                  (t (cond ((null (evaluer_liste_listes_atome
                                    liste_listes_formes t))
                            (terpri)
                            (princ "Exigences satisfaites, mais evaluation")
                            (terpri)
                            (princ "avortee, ou formules non valides! ")
                            (experience_echouee))
                           (t (terpri)
                              (princ "Experience reussie! "))))))))))))))))

;donc, si chacune des formes de liste_Echecs ne s'evalue pas a nil, sans
;erreur, l'evaluation des formes DOIT avorter
```

Exemple : les axiomes de File

```
;Nous sommes sous lisp, et les fichiers de travail ont ete charges.
;Nous travaillons sur le type File(Entier).

;Nous produisons maintenant le jeu de test de niveau 2 pour l'axiome 1 du type
;File.

(tester 1 'File 2)

;Voici cet axiome:

(1. (EGAL (enlever (ajouter q i))
          (SAS (EGAL q (file_vide)) (file_vide) (ajouter (enlever q) i))))

On y va!!

;Voici la liste de candidats deduits de cet axiome:

(((Exigences ((Preconditions nil)
              (Echecs nil)
              (Contraintes ((EGAL g0048 (file_vide))))))
  (Variables ((g0048 File 2.) (g0049 Entier)))
  (Termes (((EGAL (enlever (ajouter g0048 g0049)) (file_vide))))))
 ((Exigences ((Preconditions nil)
              (Echecs nil)
              (Contraintes ((null (EGAL g0050 (file_vide)))))))
  (Variables ((g0050 File 2.) (g0051 Entier)))
  (Termes (((EGAL (enlever (ajouter g0050 g0051))
                  (ajouter (enlever g0050) g0051)))))))


;Voici le premier de ces deux candidats totalement instancie: c'est la premiere
;experience produite par l'algorithme.

((Exigences ((Preconditions nil)
             (Echecs nil)
             (Contraintes ((EGAL (ajouter '(1. -4. 6.) (/0))
                                 (file_vide))))))
 (Variables nil)
 (Termes (((EGAL (enlever (ajouter (ajouter '(1. -4. 6.) (/0)) (/0)))
                 (file_vide))))))


;Et voici le resultat de l'evaluation de ce candidat:

--------------   1.

Contraintes non valides!


;Passons maintenant a l'axiome 2 du meme type File. Son test sera rapide...


(tester 2 'File 2)


(2. (EGAL (enlever (file_vide)) (file_vide)))
On y va!!
```

```
--------------   1.

Experience reussie!
C'est fini!

;Cet axiome est donc bien valide!

;Passons au test de l'axiome 3 du type File, au niveau 2

(tester 3 'File 2)


(3. (EGAL (premier (ajouter q i)) (SAS (EGAL q (file_vide)) i (premier q)))
    (Preconditions (premier)))
On y va!!


;Examinons simplement la liste de candidats deduite de cet axiome

(((Exigences ((Preconditions ((null (EGAL (ajouter g0059 g0060) (file_vide)))))
               (Echecs nil)
               (Contraintes ((EGAL g0059 (file_vide))))))
  (Variables ((g0059 File 2.) (g0060 Entier)))
  (Termes (((EGAL (premier (ajouter g0059 g0060)) g0060)))))
 ((Exigences ((Preconditions ((null (EGAL (ajouter g0061 g0062) (file_vide)))
                              (null (EGAL g0061 (file_vide)))))
               (Echecs nil)
               (Contraintes ((null (EGAL g0061 (file_vide)))))))
  (Variables ((g0061 File 2.) (g0062 Entier)))
  (Termes (((EGAL (premier (ajouter g0061 g0062)) (premier g0061)))))))

;Remarquons essentiellement les preconditions associees a ces deux candidats!
```