Simulation entre modèles de calcul naturel et modularité des réseaux d'automate

Pacôme Perrotin

Aix*Marseille université
Initiative d'excellence

THÈSE DE DOCTORAT

Soutenue à Aix-Marseille Université
le 12 janvier 2021 par

Pacôme PERROTIN

Simulation entre modèles de calcul naturel et modularité des réseaux d'automates

Discipline
Informatique

École doctorale
ED 184 Mathématiques et informatique

Laboratoire
Laboratoire d'informatique et systèmes (LIS)

LIS
LABORATOIRE D'INFORMATIQUE & SYSTÈMES
UMR 7020

Composition du jury

Franck Delaplace Professeur des universités, Univ. Évry	Rapporteur
Emmanuel Jeandel Professeur des universités, Univ. Lorraine	Rapporteur
Nadia Creignou Professeur des universités, Univ. Aix-Marseille	Examinatrice
Enrico Formenti Professeur des universités, Univ. Côte d'Azur	Examinateur
Loïc Paulevé Chargé de recherche, CNRS (Bordeaux)	Examinateur
Nicolas Schabanel Directeur de recherche, CNRS (Lyon)	Examinateur
Kévin Perrot Maître de conférences, Univ. Aix-Marseille	Co-directeur
Sylvain Sené Professeur des universités, Univ. Aix-Marseille	Co-directeur

Je soussigné, Pacôme Perrotin, déclare par la présente que le travail présenté dans ce manuscrit est mon propre travail, réalisé sous la direction scientifique de Sylvain Sené et de Kévin Perrot, dans le respect des principes d'honnêteté, d'intégrité et de responsabilité inhérents à la mission de recherche. Les travaux de recherche et la rédaction de ce manuscrit ont été réalisés dans le respect à la fois de la charte nationale de déontologie des métiers de la recherche et de la charte d'Aix-Marseille Université relative à la lutte contre le plagiat.

Ce travail n'a pas été précédemment soumis en France ou à l'étranger dans une version identique ou similaire à un organisme examinateur.

Fait à Marseille le 20 juillet 2020

Résumé

Nous explorons différentes généralisations concernant les modèles de calcul naturel. La plus théorique est la notion de simulation entre modèles, pour laquelle nous décrivons une série de propositions de définition, en discutant des intérêts et des failles de chacune d'elles. Nous profitons des définitions les plus prometteuses pour élargir le propos sur les possibles conséquences de la simulation en théorie de la complexité, comme la construction de nouvelles classes de complexité en proposant la substitution de la réduction polynomiale par la simulation.

Notre approche plus appliquée consiste en la généralisation des réseaux d'automates sous formes de modules, qui possèdent des entrées. Ce formalisme permet d'approcher les questions de la dynamique des réseaux d'interactions sous un nouvel angle : nous explorons son utilité en tant qu'outil modulaire propre à simuler de façon flexible de nombreux objets similaires, ainsi que l'expressivité des modules acycliques. Ceux-ci permettent la caractérisation de la dynamique des réseaux d'automates sous la forme de fonctions de sortie. Cette expressivité nous autorise la description d'un processus d'optimisation de réseaux d'automates, qui réduit certains réseaux en taille tout en conservant des attracteurs équivalents.

Abstract

We explore different generalisations about natural computation models. The most theoretical is the notion of simulation between models, for which we describe a series of proposed definitions, by discussing the interests and the flaws of each of them. We take advantage of the most promising definitions to broaden the discussion on the possible consequences of simulation in complexity theory, such as the construction of new complexity classes by proposing the substitution of polynomial reduction by simulation.

Our more applied approach consists in the generalisation of automata networks by means of modules that have inputs. This formalism makes it possible to approach the questions of the dynamics of interaction networks from a new angle : we explore its usefulness as a modular tool capable of flexibly simulating many similar objects, as well as the expressiveness of acyclic modules. These allow the characterisation of the dynamics of automata networks in the form of output functions. This expressiveness allows us to describe a process for optimising automata networks that reduces certain networks in size while retaining equivalent attractors.

Remerciements

Je souhaite remercier l'ensemble des personnes qui ont contribué à cette thèse, de près ou de loin. En tête de cette liste figurent mes deux directeurs, Kévin et Sylvain, qui me connaissent et m'on guidé depuis mon master. Je remercie également l'ensemble de mon jury et de mon comité de suivi de thèse, dont les retours ont été chaleureux et encourageants. Merci à l'ensemble des personnes qui ont participé au projet FANs et des discussions qui y ont pris place, sans lesquelles ce manuscrit aurait pris une bien autre tournure.

Je remercie l'ensemble des collègues dont j'ai fait la connaissance sur le chemin de cette thèse, ainsi qu'aux amis que j'ai fait sur les planches du théâtre. Je remercie ma famille, mes amis et mes chats, pour leur présence et leur soutient.

Table des matières

Introduction

Parmi les nombreux domaines de l'informatique, le calcul naturel se distingue par ses motivations singulières et l'horizon de ses applications et de ses sources d'inspiration. Si la très grande majorité des domaines de l'informatique n'est pas étrangère à l'application de ses résultats dans d'autres disciplines scientifiques, le calcul naturel se définit – de façon informelle – comme le regroupement des modèles et des travaux qui s'intéressent au calcul qui réside dans, ou est inspiré par, la nature. Ainsi, plus qu'une application, le calcul naturel apporte un changement fondamental de perspective, et propose une collaboration d'un genre unique entre les domaines de la science expérimentale et l'informatique, domaine du calcul.

La notion de calcul dans l'informatique tient racine dans la thèse de Chuch-Turing. Cette appellation décrit un ensemble de travaux indépendants, la thèse de Church publiée au début des années 1930 étant la plus ancienne historiquement, et la thèse de Turing publiée dans un article en 1936 étant la plus emblématique. Pris dans leur ensemble, ces travaux énoncent l'existence de machines formelles très simples, comme la machine de Turing, qui possèdent la capacité d'opérer tout calcul formel qu'un humain pourrait réaliser à l'aide d'une réserve inépuisable de papier, de crayons et de temps. Ces travaux sont le point culminant d'une recherche qui a eu pour but de déterminer si la machine artificielle était capable de reproduire le raisonnement mathématique de l'humain. Bien que la thèse de Church-Turing n'apporte aucune démonstration à cette question informelle, son énoncé stipule que tout procédé effectif de calcul est équivalent à une machine de Turing; ce qui n'a pas été réfuté depuis.

Une des origines du calcul naturel prend place dans les années 1940-1950 avec les travaux de von Neumann. Plus que de comprendre le rapport entre machine et raisonnement humain, l'ambition est désormais de comprendre le rapport entre machine et nature. von Neumann s'intéresse à la question de la création d'un système formel capable de s'auto-reproduire (NEUMANN, BURKS et al. 1966) tout en subissant des variations au cours du temps, similairement aux comportements qui permettent l'évolution du vivant. Son travail le conduit à la définition d'une machine capable de recopier sa propre description et de se reconstruire à partir de cette description. Pour décrire cette machine, von Neumann définit le premier automate cellulaire, disposant d'une grille de cellules, chacune dans un état parmi 29. Dans ces états, 17 permettent la transmission d'une information, 11 autres permettent la construction des états précédents par l'utilisation de signaux, et le dernier sert d'état de fond de la configuration.

Cet automate cellulaire inspirera la création d'une multitude d'autres (CODD 2014;

LANGTON 1984 ; LANGTON 1986 ; GARDNER 1970), toujours dans la volonté d'imiter la complexité du vivant. Un modèle particulier, la boucle de Langton, est un exemple de modèle auto-reproducteur ne disposant que de 8 états. Ce modèle sera repris et dispose aujourd'hui d'une variation capable de reproduction sexuée (OROS et NEHANIV 2007). De tous ces modèles, le jeu de la vie de Conway est devenu le plus célèbre. Malgré le fait que chacune de ses cellules n'ait que deux états possibles, vivant ou mort, et se met à jour selon une simple règle de comptage, le jeu de la vie, dans son évolution, fait preuve d'une redoutable complexité. La mise à jour d'une configuration initiale aléatoire donne rapidement vie à une faune diverse de structures animées, parmi lesquelles des vaisseaux (structures qui se déplacent dans l'espace), des oscillateurs, ou des canons (structures qui génèrent des vaisseaux périodiquement). Ces structures furent utilisées, entre autres, pour la construction de circuits booléens (WAINWRIGHT 1974) et de machines de Turing (RENDELL 2016) à l'intérieur du modèle prouvant que par sa définition simple, le jeu de la vie est capable d'un calcul universel.

L'utilisation par les automates cellulaires d'un espace régulier et de règles locales permet la modélisation de certains systèmes biologiques et physiques observés dans la nature. Dans de très nombreux domaines d'applications, le concept d'automate cellulaire a évolué en des formes différentes. Il existe ainsi des modèles quantiques (LENT, TOUGAW, POROD et al. 1993 ; ARRIGHI, FARGETTON et Z. WANG 2007) ou des modèles auto-assemblants cherchant à reproduire le comportement de l'ADN (CONNOLLY, WINFREE, SPRINGER et al. 1996). Le domaine des automates cellulaires évolue en parallèle de celui qui découle des travaux sur les réseaux de neurones (MCCULLOCH et PITTS 1943), fondation du modèle des réseaux d'automates permettant notamment la modélisation du comportement de réseaux de régulation génétique (KAUFFMAN 1969 ; THOMAS 1973 ; BERNOT, GUESPIN-MICHEL, COMET et al. 2003). De plus, de nombreux modèles sont étudiés pour leur faculté à présenter une grande complexité de comportement, comme les modèles de piles de sable (BAK, TANG et WIESENFELD 1987) ou les pavages (GARDNER 1977 ; H. WANG 1961 ; BERGER 1966). Tout ces modèles sont communément appellés modèles de calcul : le domaine du calcul naturel englobe ces travaux qui cherchent à comprendre la complexité de modèles de calcul pourtant simplement définis et souvent inspirés de phénomènes complexes observés en biologie, chimie, physique, économie, sociologie et autres. Malgré cet horizon large d'applications, d'ambitions et d'inspirations, il existe des notions intuitives qui restent largement partagées dans le calcul naturel, à l'instar de la notion particulièrement intéressante que constitue la simulation.

Lorsqu'un modèle simule un autre modèle, cela signifie informellement que le modèle simulateur est capable de reproduire le modèle simulé dans son entièreté. La simulation est presque toujours opérée par la reconstruction du modèle simulé à l'intérieur du modèle simulateur, et cela produit presque toujours des illustrations intuitives, comme par exemple pour la simulation de tout circuit booléen par la fourmi de Langton (GAJARDO, MOREIRA et GOLES 2002) ou la simulation d'une machine de Turing par la règle 110 (COOK 2004). Ces illustrations, qui confèrent un sens intuitif à la construction impliquée dans une simulation, permettent au lecteur, même oc-

casionnel, de comprendre intuitivement les conséquences de la simulation sur les modèles étudiés, et ce malgré les très nombreuses définitions de modèle de calcul qui parcourent le calcul naturel, ainsi que les nombreuses définitions de simulation. En effet, malgré son attrait presque universel, la simulation ne possède aucune définition générale qui s'applique à l'ensemble des modèles de calcul naturel : il y a une grande disparité entre cette absence d'uniformité formelle et la cohérence des intuitions qui poussent l'utilisation et la représentation de la simulation dans ces travaux.

Il existe dans ce contexte une occasion unique de travailler à la généralisation d'une notion qui touche profondément aux idées qui ont initié le domaine du calcul naturel et continuent de le motiver. Si les questions originelles de la capacité des machines de présenter des comportements complexes sont satisfaites, il réside dans le domaine du calcul naturel la question profonde d'identifier les fondements de la complexité qui semble émerger de modèles de calcul simples, et qui nous rappelle si fortement la complexité de la nature. Le mot complexe fait partie, avec le mot calcul, de ce vocabulaire pour lequel le calcul naturel propose une intuition forte mais échoue à proposer une définition formelle convainquante. La simulation, à défaut de définir directement la complexité ou le calcul, est une notion largement employée d'une façon à la fois formelle – mais propre à des modèles donnés – et intuitive. Si formaliser la simulation et en proposer une définition générale est un défi dont l'ambition dépasse celle de ce manuscrit, il représente à nos yeux la prochaine grande question du domaine du calcul naturel, et il est certain que la réussite – ou l'échec – d'une tentative sérieuse d'y répondre sera d'une grande valeur scientifique.

Ce manuscrit est la collection d'un travail de thèse qui ne s'adresse que partiellement à la question de la simulation. Le chapitre 2 fait l'exposé d'une sélection de formalisations de la simulation et tente de poursuivre les intuitions qu'elles inspirent pour en proposer une définition générale. Ces intuitions donnent en premier lieu la simulation par forme, qui tente de concrétiser l'idée de remise à l'échelle qui se trouve être au cœur de nombreuses définitions intuitives et pratique de la simulation. Une seconde définition, fortement basée sur la définition de la bisimulation entre systèmes de transition d'états, semble plus prometteuse et permet une mesure en complexité de la simulation entre modèles.

Cette mesure en complexité est exploitée au chapitre 3 avec la proposition d'une extension des classes de la théorie de la complexité dans le domaine des systèmes de transition d'états. Cette extension se limite à l'étude du cas des classes P, NP et coNP, pour lesquelles le plongement dans l'espace des systèmes de transition d'états préserve leur structure et pose des questions subtiles sur la nature de ces classes et de leur formalisation. Ce chapitre discute également plus profondément des intuitions qui guident notre compréhension de la simulation, du calcul et de l'émergence.

Les chapitres 4, 5 et 6 regroupent nos travaux sur le modèle des réseaux d'automates. Ces réseaux sont une généralisation des automates cellulaires, dans laquelle certaines propriétés qui les caractérisent – la régularité spatiale, l'uniformité, le synchronisme – ne sont pas conservées. On obtient un modèle général propre à simuler n'importe quelle structure faite d'agents simples, nommés automates. Loin de permettre une

compréhension facile des automates cellulaires, les réseaux d'automates font preuve de leurs propres sources de complexité et d'application uniques. Leur formalisme est développé dans le détail au chapitre 4.

Comprendre et caractériser la dynamique des réseaux d'automates est un projet ambitieux et complexe. Notre contribution consiste en une généralisation des réseaux d'automates en un nouveau formalisme, les modules, développé au chapitre 5. En plus des automates qui constituent les réseaux susnommés, les modules disposent d'entrées extérieures à l'évaluation arbitraire qui influencent leurs automates. Cet apport complexifie un modèle de calcul dont la dynamique échappe déjà à toute caractérisation. L'intuition est la suivante : les modules permettent de modéliser le comportement d'un sous-réseau d'automates faisant partie d'un réseau plus large. Les entrées, dans ce contexte, représentent l'influence du reste du réseau sur le sous-réseau considéré. Cette intuition est mise en pratique dans l'expression d'un méta-théorème portant sur la simulation, qui utilise les modules et leurs entrées pour permettre une définition de simulation locale. En plus de ces entrées, les modules peuvent être combinés par des branchements qui opèrent comme des soudures et permettent le remplacement de l'influence d'une entrée par l'influence d'un automate, que ce soit l'automate du module considéré ou d'un autre module. Ainsi, plutôt que d'étudier un réseau de grande taille pour laquel le calcul de la dynamique est complexe, on pourra découper ce réseau en modules et obtenir une série d'objets plus simples à analyser et prédire.

Puisque les modules sont une généralisation d'un modèle de calcul déjà très général, nous simplifions notre recherche en étudiant au chapitre 6 une restriction des modules nommée modules acycliques ; ils ont la particularité de posséder un graphe d'interaction acyclique. Cette propriété fait des modules acycliques des objets suffisament simples pour permettre une caractérisation de leur calcul, mais assez complexe pour permettre de prédire la dynamique de réseaux d'automates. Un résultat en particulier permet la caractérisation du calcul d'un module acyclique sous la forme de fonctions de sortie, qui sont des fonctions qui prennent en entrée l'historique des valeurs des entrées du module, et prédisent asymptotiquement la valeur de tout automate du réseau. Ces fonctions de sortie possèdent une forte relation avec la dynamique de ces modules. Pour deux modules donnés aux fonctions de sortie équivalentes au renommage des entrées et des automates près et lorsque ces deux modules sont branchés récursivement de façon équivalente, on montre que les dynamiques limites des réseaux obtenus sont isomorphes. Ce résultat permet la formalisation d'une méthode d'optimisation de réseaux d'automates, qui consiste en la transformation d'un tel réseau en module acyclique, en l'extraction des fonctions de sortie de ce module, de la génération d'un module optimal en taille qui possède les même fonctions de sortie, et enfin de la recomposition de ce module optimal en un réseau d'automates. Le réseau ainsi obtenu est assuré d'avoir une dynamique limite isomorphe au réseau initial, et son nombre d'automates est plus petit ou égal au nombre d'automates du réseau initial. Nous considérons le cas particulier des modules acycliques ne disposant que d'une seule entrée, nommés modules 1-pour-1, et pour lesquels cette méthode

d'optimisation se trouve être particulièrement efficace. Ces modules qui semblent triviaux englobent plusieurs familles de réseaux, comme les fleurs d'automates et les registres à décalage à rétroaction linéaire.

En addition à tous ces éléments, le chapitre 1 propose une introduction à toutes les notions nécessaires à une lecture sereine de ce manuscrit.

1 Définitions et notations de base

Ce chapitre détaille les définitions de base qui sont utilisées dans ce manuscrit. Le lecteur est convié à y retourner à tout instant pour clarifier notre intention autour de ces définitions. Il dispose également d'un index alphabétique en fin de manuscrit qui dispose de pointeurs vers toutes les définitions explorées entre ces pages.

1.1 Ensembles et séquences

Pour S un ensemble, le cardinal de S, noté $|S|$, est le nombre d'éléments que contient S. L'ensemble des parties de S, noté $\wp(S)$, est l'ensemble $\wp(S) = \{S' | \ S' \subseteq S\}$.

Exemple 1. *Soit* $S = \{a, b, c\}$ *un ensemble. On observe* $|S| = 3$ *et* $\wp(S) = \{\emptyset, \{a\}, \{b\}, \{c\}, \{a, b\}, \{b, c\}, \{a, c\}, \{a, b, c\}\}$.

Une séquence est un ensemble ordonné d'éléments, que l'on note $P = (p_1, p_2, \ldots)$. Pour Λ un ensemble, l'ensemble des séquences finies dont les éléments sont dans Λ est noté Λ^*. Une séquence peut être finie ou infinie. Dans le cas d'une séquence finie, la taille de la séquence est notée $|P|$. Pour tout nombre $k > 0$, le k-ième élément de P est également noté P_k.

Soient P et P' deux séquences d'ensembles. On définit l'union de P et P', notée $P \cup P'$, comme la séquence composée de l'union des ensembles dans l'ordre des séquences originelles. Pour tout $k > 0$, la séquence est définie par

$$(P \cup P')_k = \begin{cases} P_k \cup P'_k & \text{si } P_k \text{ et } P'_k \text{ sont définis} \\ P_k & \text{si seulement } P_k \text{ est défini} \\ P'_k & \text{si seulement } P'_k \text{ est défini} \end{cases}.$$

Exemple 2. *Pour* $P = (\{1\}, \{2\}, \{3\})$ *et* $Q = (\{a\}, \{b\}, \{c\}, \{d\})$, $P \cup Q = (\{1, a\}, \{2, b\}, \{3, c\}, \{d\})$.

La séquence P^k, pour k un nombre entier, est la séquence construite comme la répétition de la séquence P un nombre k de fois.

Exemple 3. *Si* $P = (1, 2, 3)$, *alors* $P^3 = (1, 2, 3, 1, 2, 3, 1, 2, 3)$.

1.2 Fonctions

Une fonction f est dite *partielle* de A vers B s'il existe un sous-ensemble $A' \subseteq A$ tel que f est définie sur $A' \to B$. On note alors $f : A \nrightarrow B$. S'il n'existe pas de tel sous-ensemble $A' \subsetneq A$, f est dite *totale*.

Exemple 4. *Soit $f : \mathbb{R} \to \mathbb{R}$ qui, à tout nombre réel a, associe la valeur $f(a) = \frac{a}{2}$. La fonction f est correctement définie pour toute valeur de $\mathbb{R}$, elle est donc totale. Soit $g : \mathbb{R} \nrightarrow \mathbb{R}$ la fonction pour tout nombre réel a associe la valeur $g(a) = \frac{1}{a}$. Par définition, g est une fonction totale sur $\mathbb{R} \setminus \{0\} \to \mathbb{R}$, ce qui fait de g une fonction partielle.*

Pour une fonction $f : A \to B$, l'ensemble A est le *domaine* de f, noté $\mathrm{dom}(f)$. L'ensemble B est le *co-domaine* de f, noté $\mathrm{codom}(f)$.

Pour une fonction $f : A \to B$, on nomme image de f l'ensemble défini par $\mathrm{img}(f) = \{f(a) \mid a \in A\}$. L'image d'une fonction diffère de son co-domaine dès que la fonction n'est pas surjective, c'est-à-dire qu'il existe au moins une valeur $b \in B$ telle qu'il n'existe aucun $a \in A$ tel que $f(a) = b$.

Exemple 5. *Soit $f : \mathbb{N} \to \mathbb{N}$ la fonction qui à tout entier n associe $f(n) = 2n$. L'ensemble des valeurs retournées par la fonction f, $img(f)$, est l'ensemble des entiers naturels pairs.*

Soient f et g deux fonctions définies sur $f : A \to B$ et $g : B \to C$. La *composition* de g et f, notée $g \circ f$, est la fonction définie telle que $g \circ f : A \to C$ qui vérifie $g \circ f(x) = g(f(x))$.

Exemple 6. *Soit $f, g : \mathbb{R} \to \mathbb{R}$ deux fonctions définies telles que $f(x) = \frac{x}{2}$ et $g(x) = x + 1$. Par définition, $(f \circ g)(x) = \frac{x+1}{2}$, et $(g \circ f)(x) = \frac{x}{2} + 1$.*

Soit f une fonction définie sur $f : A \to B$ et A' un sous-ensemble de A. La *réduction* de f à A' est la fonction notée $f|_{A'}$ définie sur $f|_{A'} : A' \to B$ telle que $f|_{A'}(x) = f(x)$.

Pour tout ensemble A, on définit la *fonction identité*, dénotée id, comme la fonction définie sur $A \to A$ telle que $\mathsf{id}(a) = a$ pour tout $a \in A$.

Pour f une fonction définie sur $f : A \to A$ et k un nombre entier, on définit l'exponentiation de f par k, notée f^k, comme la fonction définie par les équations récursives

$$f^0 = \mathsf{id}$$
$$f^{k+1} = f \circ f^k.$$

Exemple 7. *Soit $f : \mathbb{N} \to \mathbb{N}$ la fonction qui vérifie $f(n) = n+1$. Par définition, la fonction f^k est la fonction définie sur les mêmes ensembles qui vérifie $f^k(n) = n + k$.*

1.3 Vecteurs

Un *vecteur* est une collection de valeurs indicées par un certain ensemble. Pour A l'ensemble d'indices et B l'ensemble de valeurs, l'ensemble des vecteurs qui associent

une valeur dans B aux indices dans A est noté B^A. Pour $x \in B^A$ un certain vecteur et $a \in A$, la valeur associée à l'indice a dans x est notée x_a.

Dès lors qu'il existe un ordre implicite sur l'ensemble des indices d'un vecteur, nous introduisons la notation alternative des valeurs de ce vecteur comme un mot composés des valeurs du vecteur dans l'ordre croissant de leurs indices.

Exemple 8. *Soit $A = \{a, b, c, d\}$ et $B = \{0, 1, 2\}$. Soit v un vecteur défini sur les indices A et avec valeurs dans B tel que $v_a = 0$, $v_b = 2$, $v_c = 0$ et $v_d = 1$. La valeur de ce vecteur peut être également notée* 0201.

Pour x un vecteur dans B^A et y un vecteur dans $B^{A'}$, tels que $A \cap A' = \emptyset$, on définit la concaténation de x et y, notée $x \cdot y$, comme le vecteur dans $B^{A \cup A'}$ tel que

$$(x \cdot y)_a = \begin{cases} x_a & \text{si } a \in A \\ y_a & \text{si } a \in A' \end{cases}.$$

Exemple 9. *Soient $A = \{a, b\}, A' = \{c, d\}, B = \{0, 1, 2\}$ trois ensembles, avec $v =$ 02 un vecteur défini dans B^A et $v' =$ 01 un vecteur défini dans $B^{A'}$. La concaténation de v et v' est égale à $v \cdot v' =$ 0201.*

Alternativement, nous définissons la fonction concat qui, pour tout ensemble de vecteurs, retourne la concaténation de cet ensemble :

$$\text{concat}(\{v_1, v_2, \ldots, v_n\}) = v_1 \cdot v_2 \cdot \ldots \cdot v_n.$$

Sans perte de généralité, on peut, à tout moment, considérer un vecteur $x \in B^A$ comme une fonction $f : A \to B$ qui associe une valeur à chacun de ses indices. En ces termes, nous étendons certaines notations spécifiques aux fonctions sur les vecteurs.

Pour x un vecteur dans B^A et $f : C \to A$ une fonction, on définit la *composition* de x et f, notée $x \circ f$, le vecteur définit dans B^C qui vérifie $(x \circ f)_c = x_{f(c)}$, pour tout c dans C.

Exemple 10. *Soit $v =$ 0201 un vecteur défini sur B^A, pour $B = \{0, 1, 2\}$ et $A = \{a, b, c, d\}$. Soit $f : A \to A$ la fonction définie par $f(a) = d$, $f(b) = c$, $f(c) = b$ et $f(d) = a$. On observe $v \circ f =$ 1020.*

Pour x un vecteur dans B^A et A' un sous-ensemble de A, on définit la projection de x dans $B^{A'}$, notée $x|_{A'}$, comme le vecteur qui vérifie $(x|_{A'})_a = x_a$ pour tout a dans A'.

Exemple 11. *Soit $v =$ 0201 un vecteur défini sur B^A, pour $B = \{0, 1, 2\}$ et $A = \{a, b, c, d\}$. Soit $A' = \{b, c\}$. On observe $v|_{A'} =$ 20. On note que toute projection peut être alternativement définie comme une composition. Dans notre exemple, on prend $f : \{b, c\} \to A$ la fonction qui vérifie $f(b) = b$ et $f(c) = c$. Par définition, $v \circ f = v|_{A'} =$ 20.*

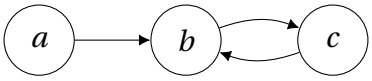

FIGURE 1.1 – Illustration du graphe G défini dans l'exemple 12.

1.4 Graphes

Un *graphe orienté* $G = (S, A)$ est défini par un ensemble de sommets S et un ensemble d'arcs $A \subseteq S \times S$. Les ensembles S et A sont respectivement notés $V(G)$ et $E(G)$. Dans le reste de ce manuscrit, le terme "graphe" sera régulièrement utilisé pour décrire un graphe orienté.

Exemple 12. *Soient $S = \{a, b, c\}$ un ensemble de sommets et $A = \{(a, b), (b, c), (c, b)\}$ un ensemble d'arcs. Le graphe orienté G qui vérifie $V(G) = S$ et $E(G) = A$ est représenté en figure 1.1.*

L'*arité entrante* d'un sommet s dans un graphe $G = (S, A)$ est définie par le nombre d'arêtes uniques (s', s) dans A. L'*arité sortante* d'un sommet s dans un graphe $G = (S, A)$ est définie par le nombre d'arêtes uniques (s, s') dans A. L'*arité* d'un sommet est la somme de son arité entrante et de son arité sortante.

Pour $G = (S, A)$ et $G' = (S', A')$ deux graphes, le graphe G' inclut le graphe G', noté $G \subseteq G'$, si et seulement si $S \subseteq S'$ et $A \subseteq A'$.

On appelle *chemin* toute séquence de nœuds $(s_1, s_2, \ldots)$ telle que $(s_i, s_{i+1}) \in A$ pour tout i. Un *cycle* est un chemin $(s_1, s_2, \ldots s_k)$ qui vérifie $s_1 = s_k$.

Un graphe G est *acyclique* s'il ne dispose d'aucun cycle. Dans un graphe acyclique, les sommets d'arité sortante 0 sont appelés les puits, et les sommets d'arité entrante 0 sont appelés les sources.

Pour $G = (S, A)$ un graphe, G est un *arbre* si et seulement si pour toute paire de sommets (s, s'), il n'existe au plus qu'un seul chemin possible pour aller de s à s' dans G. Les puits d'un arbre sont appelés les feuilles, et les sources sont appelées les racines.

Un *graphe étiqueté* est défini par un ensemble de sommets S, un ensemble d'arêtes A et un ensemble d'étiquettes E, tels que $A \subseteq S \times E \times S$. Une arête (s, e, s') signifie que l'étiquette e décore l'arête allant de s vers s'.

Un graphe $G = (S, A)$ est *fortement connexe* si et seulement si, pour toute paire s, s' de sommets distincts, il existe un chemin de s vers s' et un chemin de s' vers s. De façon similaire, une *composante fortement connexe* de G est un ensemble maximal $S' \subseteq S$ tel que pour toute paire s, s' de sommets distincts dans S', il existe un chemin de s vers s' et de s' vers s parmi les sommets S'.

On nomme *composante fortement connexe terminale* d'un graphe toute composante fortement connexe dont il est impossible de sortir. Formellement, pour $G = (S, A)$ et S' un sous-ensemble de S, S' décrit une composante fortement connexe terminale si et seulement si S' est une composante fortement connexe et si pour tout $s, s' \in S$, si $s \in S'$ et $(s, s') \in A$, alors $s' \in S'$.

1.5 Fonctions booléennes

L'ensemble des nombres *booléens* est noté $\mathbb{B}$ et défini par $\mathbb{B} = \{0,1\}$. Une *fonction booléenne* est une fonction définie de $\mathbb{B}^A \to \mathbb{B}$, pour A un certain ensemble. Pour $f : \mathbb{B}^A \to \mathbb{B}$, l'arité de f est le cardinal de A.

Une *porte booléenne* est une fonction parmi $\neg$, $\vee$ et $\wedge$. La porte $\neg$, aussi appelée "non logique", est d'arité un et est définie par $\neg 0 = 1$ et $\neg 1 = 0$. Les portes $\vee$ et $\wedge$ sont d'arité deux : la porte $\vee$, aussi appelée "ou logique", vaut 1 si et seulement si au moins une de ses deux entrées est évaluée à 1. la porte $\wedge$, aussi appelée "et logique", vaut 1 si et seulement si ses deux entrées sont évaluées à 1.

Les portes booléennes sont combinées pour former des graphes qui permettent le calcul de fonctions booléennes. Ces graphes sont acycliques, et ne possèdent qu'un seul puit. Les sources d'un tel graphe représentent les variables de la fonction booléenne, et chaque autre sommet est associé à une porte booléenne qui calcule sa valeur selon la valeur des nœuds incidents. Il est important que l'arité entrante de chaque sommet qui dispose d'une porte corresponde à l'arité de la porte : arité entrante 1 pour une porte non, et arité 2 pour une porte ou, ou et. Pour exécuter la fonction booléenne d'un tel graphe, on note la valeur de chaque variable, et on remonte le long des arcs du graphe jusqu'au puit du réseau, en exécutant les portes rencontrées sur le chemin. La valeur trouvée au puit du réseau correspond à l'évaluation de la fonction encodée par le réseau. Dans le cas général, ces objets sont appelés *circuits booléens*.

Exemple 13. *Soit $S = \{a, b, c, p1, p2\}$ un ensemble de nœuds, et les arêtes $A = \{(a, p1), (b, p1), (c, p2), (p1, p2)\}$. On associe à $p1$ la porte logique ou, et à $p2$ la porte logique et. Le circuit booléen C associé encode une fonction booléenne f d'arité 3 dont la table de vérité est*

a	b	c	f
0	0	0	0
0	0	1	0
0	1	0	0
0	1	1	1
1	0	0	0
1	0	1	1
1	1	0	0
1	1	1	1

Le graphe du circuit défini dans l'exemple 13 est représenté en figure 1.2.

Lorsqu'un tel graphe possède la propriété supplémentaire d'être un arbre, on l'appelle une *formule booléenne*. La structure d'une formule booléenne permet de la représenter sous la forme d'une formule mathématique, en représentant la structure de l'arbre grâce à des parenthèses.

Exemple 14. *On considère le circuit booléen C décrit dans l'exemple 13. Le graphe du circuit C possède la propriété d'être un arbre, et est donc également une formule booléenne. Son écriture sous forme de formule est $(a \vee b) \wedge c$.*

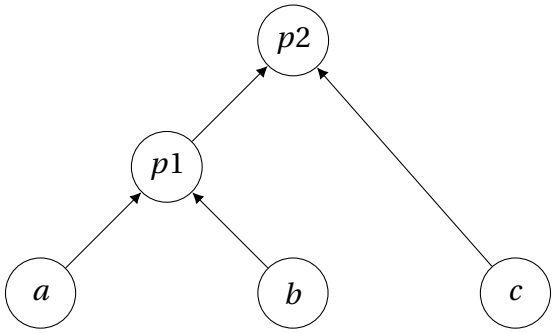

FIGURE 1.2 – Graphe du circuit booléen défini dans l'exemple 13.

1.6 Machines de Turing

Les machines de Turing sont un modèle de calcul que l'on peut représenter comme un robot qui avance et recule sur un ruban infini d'information. Le robot dispose d'une tête de lecture et écriture, d'un état et d'une table de transition. Cette table contient une entrée pour chaque état et chaque valeur de lecture possible ; pour chacune de ces paires, cette table retourne une valeur à écrire, un déplacement optionnel et un nouvel état pour la machine de Turing. Mettre à jour la machine repose sur la lecture de cette table et l'application de son résultat. Ce calcul peut ne jamais s'arrêter ; cependant on définit souvent des états spéciaux, dit finaux, qui une fois établis vérouillent la machine dans une position et préviennent toute autre modification. C'est une convention pour signifier la fin du calcul d'une machine de Turing.

On fait la distinction entre les machines de Turing déterministes et les machines de Turing non déterministes sur le critère qu'une machine de Turing déterminisite dispose toujours d'un et un seul choix en toute circonstance. Une machine de Turing non déterminisite peut au contraire disposer d'une table de transition qui définit au moins deux possibles résultats pour les mêmes circonstances.

1.6.1 Cas déterministe

Formellement, une machine de Turing déterministe se définit comme un quintuplet $(S, \Lambda, s_0, \delta, F)$, où S est un ensemble fini d'états, Λ est l'alphabet des symboles du ruban, s_0 est un état de S dit initial, $\delta : S \times \Lambda \to S \times \Lambda \times \{-1, 0, +1\}$ est la fonction qui encode la table de transition, et F décrit le sous-ensemble des états de la machine dits finaux. Le ruban de la machine de Turing est représenté par un vecteur bi-infini $R \in \Lambda^{\mathbb{Z}}$.

Pour mettre à jour une telle machine $T = (S, \Lambda, s_0, \delta, F)$ depuis un état s sur le ruban $R \in \Lambda^{\mathbb{Z}}$, et telle que la position de la tête soit actuellement un certain $z \in \mathbb{Z}$, nous calculons $(s', y, p) = \delta(s, R_z)$. Il résulte de cette mise à jour que la machine T est maintenant à l'état s' et sa tête à la position $z + p$, et le nouveau ruban R' est défini en tout point égal à R excepté $R'_z = y$. Dans le cas particulier où l'état actuel de la machine est dans F, la machine n'est pas mise à jour.

Exemple 15. *Soit* $T = (\{a,b\}, \mathbb{B}, a, \delta, \emptyset)$ *la machine de Turing telle que*

$$\begin{aligned}\delta(a,0) &= (b,1,+1)\\ \delta(a,1) &= (a,0,-1)\\ \delta(b,0) &= (b,1,-1)\\ \delta(b,1) &= (a,0,+1).\end{aligned}$$

On observe l'exécution périodique de T suivante :

état a	$\ldots 000\check{0}000\ldots$
état b	$\ldots 0001\check{0}00\ldots$
état b	$\ldots 000\check{1}100\ldots$
état a	$\ldots 0000\check{1}00\ldots$
état a	$\ldots 000\check{0}000\ldots$

Chaque ligne correspond à une étape de la mise à jour de la machine T. Dans cette représentation, l'état de la machine de Turing est indiqué sur la gauche, et la configuration d'un segment du ruban bi-infini est indiquée sur la droite. Les notations $\check{0}$ *et* $\check{1}$ *précisent la position de la tête de lecture.*

1.6.2 Cas non déterminisite

Une machine de Turing non déterministe est un quintuplet $(S, \Lambda, s_0, \Delta, F)$, où S, Λ, s_0 et F sont définis de la même façon que pour une machine déterministe. Ici, Δ est défini comme un sous-ensemble de $S \times \Lambda \times S \times \Lambda \times \{-1, 0, +1\}$, tel que pour toute paire (s, x), il existe au moins un triplet (s', y, p) tel que $(s, x, s', y, p) \in \Delta$.

Ainsi, pour tout quintuplet $(s, x, s', y, p) \in \Delta$, la machine a la possibilité de passer de l'état s à l'état s' lorsqu'elle lit la valeur x, en entraînant l'écriture de l'état y et le déplacement p.

Mettre à jour une machine de Turing non déterministe à partir d'une valeur de lecture x et d'un état s repose sur la sélection d'un quintuplet $(s, x, s', y, p) \in \Delta$. Lorsque plusieurs tels quintuplets existent, le calcul est non déterministe ; plutôt que de disposer d'une exécution possible, la machine possède plusieurs possibilités. On considèrera en général chaque possibilité comme dessinant un arbre de configurations dont la racine est la configuration initiale de la machine, telle que chaque nœud de l'arbre dispose d'autant de fils qu'il y a de possibles mises à jour de la configuration qui correspond au nœud. Ainsi, les feuilles de l'arbre correspondent aux différentes façons dont la machine peut terminer son calcul à partir de cette configuration initiale.

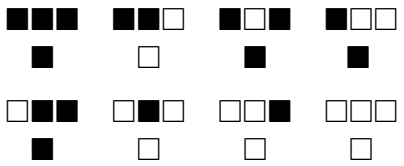

FIGURE 1.3 – Table de vérité de la fonction booléenne qui encode l'automate cellulaire élémentaire 184. Ce nombre est encodé par le nombre binaire obtenu par la concaténation de la réponse de la fonction à chaque configuration binaire de longueur 3, dans l'ordre décroissant. Dans notre exemple, $184_{10} = 10111000_2$.

1.7 Automates cellulaires élémentaires

Les automates cellulaires disposent d'un ensemble régulier de cellules, comme une grille ou une ligne, et chacune de ces cellules contient une valeur. On met à jour un automate cellulaire en appliquant à chaque cellule une fonction locale qui prend en entrée la valeur de la cellule et de ses voisins. Les automates cellulaires élémentaires sont un ensemble de 256 automates cellulaires parmi les plus simples qu'il est possible de définir : ils opèrent sur des configurations en forme de ligne et sur un alphabet binaire, et considèrent une définition de voisinage très simple : les voisins d'une cellule sont elle-même, sa voisine de gauche, et sa voisine de droite.

Ainsi, la fonction locale d'une cellule d'un automate cellulaire élémentaire est une fonction qui prend en entrée trois valeurs booléennes et retourne une valeur booléenne. Il n'existe que $2^{2^3} = 256$ de ces fonctions, ce qui invite la convention de nommer un automate cellulaire élémentaire par le numéro qui encode la fonction locale qui le définit (WOLFRAM 1984).

Exemple 16. *On considère l'automate cellulaire élémentaire* 184, *dont la table de vérité de la fonction locale est représentée en figure 1.3. Soit x une configuration représentée par un vecteur* 01100010111010. *On suppose que cette configuration forme une boucle de façon à ce que le voisin de gauche de la cellule plus à gauche soit la cellule la plus à droite, et vice versa. La mise à jour de la règle* 184 *sur cette configuration donne la configuration* 01010001110101.

Malgré la grande simplicité de définition des automates cellulaires élémentaires, l'automate cellulaire élémentaire 110, aussi connu sous le nom règle 110, est connu pour être capable de calcul universel. Ce calcul prend place grâce à l'interaction de planeurs dans l'évolution de la configuration qui permettent un transfert d'information. La table de vérité de cet automate est illustrée en figure 1.4.

Le calcul d'un automate cellulaire élémentaire repose sur la mise à jour successive d'une configuration initiale. La séquence des configurations ainsi obtenues s'appelle un diagrame espace-temps. L'illustration d'une exécution de la règle 110 est faite en figure 1.5.

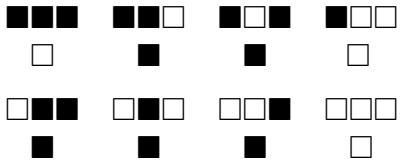

FIGURE 1.4 – Table de vérité de la fonction booléenne qui encode l'automate cellulaire élémentaire 110.

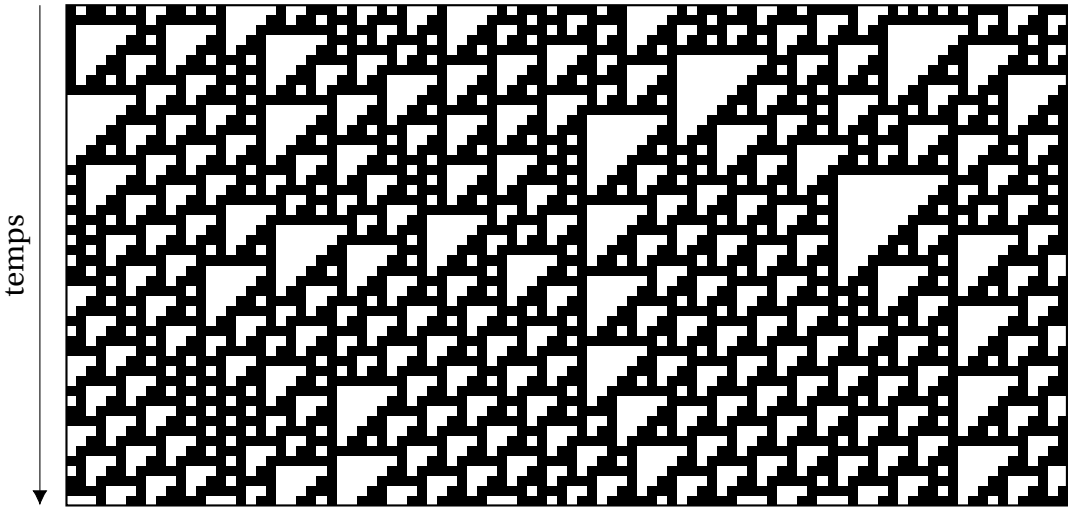

FIGURE 1.5 – Diagramme espace-temps d'une exécution de la règle 110. L'exécution est représentée sous la forme d'une grille, dans laquelle chaque cellule noire correspond à la valeur 1 et chaque cellule blanche correspond à la valeur 0. Chaque ligne correspond à une configuration. La configuration initiale de l'automate correspond à la première ligne de la grille. La mise à jour de cette configuration est représentée juste en dessous : la lecture d'une telle exécution se fait de haut en bas.

Une configuration d'un automate cellulaire peut être finie ou infinie ; dans le contexte de ce manuscrit, l'information utile de chaque configuration sera limitée. Par conséquent, on considèrera les configurations des automates cellulaires soit comme des configurations finies et circulaires, c'est-à-dire dont la cellule la plus à gauche est définie comme la voisine de droite de la cellule la plus à droite, et vice versa ; soit comme infinie et redondante, c'est-à-dire comme la répétition infinie d'un segment donné. Dans les deux cas, la dynamique observée est moralement la même, et nous illustrerons les deux cas de façon équivalente.

1.8 Complexité

La théorie de la complexité procure un ensemble d'outils qui permet la mesure de la difficulté d'une question ou de l'efficacité d'un algorithme. Elle est utilisée dans pratiquement tous les domaines de l'informatique théorique. Nous définissons ici les bases qui permettent la lecture sereine de ce manuscrit. Cette théorie repose sur la définition et la catégorisation de problèmes.

1.8.1 Problèmes de décision

Un problème de décision définit un ensemble infini d'instances, chacune d'entre elles étant une instance positive ou négative. Une instance peut être vue comme une question ; le signe de l'instance représente la réponse à cette question. Un problème de décision se définit de la façon suivante :

▶ Problème d'évaluation d'un circuit booléen
Entrée: Un circuit C, encodé comme un graphe.
Question: L'évaluation de la racine du réseau C donne-t-elle 1 ?

La définition d'un problème de décision repose ainsi sur trois éléments : un nom, une famille d'instances (dans cet exemple, nous définissons une instance pour chaque circuit C), et une question pour chacune de ces instances. Résoudre ce problème repose sur la capacité à résoudre cette question pour toute instance.

1.8.2 Classes de complexité

Pour classifier l'infinité des problèmes de décision possibles, la théorie de la complexité propose un ensemble de classes, qui déterminent le type de machine et le temps nécessaire à résoudre l'ensemble des questions posées par le problème. Parmi les plus célèbres, la classe P contient tous les problèmes qui peuvent être résolus avec une machine de Turing déterministe en temps polynomial dans le pire des cas. Cela signifie qu'il existe un polynôme g et une machine de Turing déterministe T tels que lorsque T est initiée sur le nombre k qui correspond à une instance du problème, cette machine répond correctement par "oui" ou "non" par le biais d'états finaux spécialisés après au plus $g(|k|)$ étapes de temps, où $|k|$ est le nombre de bits nécessaires pour représenter k.

Parmi d'autres classes célèbres, nous pouvons citer NP, définie comme la classe de problèmes dont la solution peut être vérifiée en temps polynomial avec une machine de Turing déterministe dans le pire des cas, c'est-à-dire que pour toute instance positive du problème il existe un p-prédicat de taille polynomiale qui permet de vérifier que l'instance est positive. Définie comme la classe duale de NP, coNP est la classe qui contient les problèmes dont les instances peuvent être vérifiées négatives en temps polynomial par une machine de Turing déterministe dans le pire des cas, c'est-à-dire que, pour toute instance négative, il existe un p-prédicat de taille polynomiale

qui permet de vérifier que l'instance est négative. Dans le cas de la classe NP, le p-prédicat sert généralement à donner la solution ; dans le cas de la classe coNP, ce p-prédicat représente généralement une sorte de contre-exemple à l'instance.

La classe FPT (pour *Fixed Parameter Tracktable*, DOWNEY, FELLOWS et STEGE 1999) contient les problèmes de la forme (x, k), pour k un paramètre, qui peuvent être résolus par une machine Turing déterministe en au plus $f(k)g(|x|)$ étapes de temps, où g est un polynôme et f une fonction arbitraire. Nous utilisons cette classe pour illustrer quels aspects d'une instance d'un problème donné contribuent le plus fortement à sa complexité.

On peut également citer la classe EXP qui contient les problèmes qui peuvent être résolus avec une machine de Turing qui dispose d'un temps exponentiel en la taille de l'entrée dans le pire des cas.

1.8.3 Oracles

Une autre façon de définir des classes de complexité est par l'utilisation d'oracles. La notation coNP^{NP} décrit la classe de problèmes qui sont dans coNP lorsque l'on dispose d'un oracle NP. Un oracle est une machine qui permet la résolution immédiate d'un problème de décision. Un problème est dans NP^{NP}, par exemple, si ce problème peut être résolu par une machine de Turing non déterministe qui dispose de la capacité d'appeller un nombre polynomial d'oracles NP. Sous l'hypothèse que la classe P est différente de la classe NP, cette notation d'oracles, utilisée sur les classes P, NP et coNP, permet la définition d'une famille infinie de classes de complexité, appelée la hiérarchie polynomiale.

1.8.4 Réduction

La notion de réduction permet de montrer qu'un problème peut se réduire en tout point à un autre. En particulier, la réduction polynomiale se définit de la façon suivante : une réduction polynomiale depuis un problème A vers un problème B est une fonction f définie depuis les instances de A vers les instances de B calculable par une machine de Turing en temps polynomial, de façon à ce que x est une instance positive (resp. négative) de A, si et seulement si $f(x)$ est une instance positive (resp. négative) de B. On note cette réduction $A \leq_p B$. Prouver une telle réduction polynomiale permet de montrer que le problème A est au moins aussi facile que le problème B, puisque l'on peut maintenant résoudre A en résolvant le problème B avec le prix ajouté d'une fonction calculable en temps polynomial. Cette réduction montre également que le problème B est au moins aussi difficile que le problème A ; en effet, résoudre toutes les instances de B demande en particulier de savoir résoudre chaque instance de A. La réduction polynomiale est le principal outil qui permet de prouver l'appartenance ou la non appartenance d'un problème à la classe NP et à toutes les classes plus larges que NP dans la hiérarchie polynomiale. Si tout les problèmes d'une classe donnée peuvent être réduits à un problème particulier, alors ce problème est un problème

difficile de cette classe. Si un problème est à la fois un problème difficile d'une classe et un problème de cette classe, alors ce problème est dit complet pour cette classe. Les problèmes complets sont les représentants des problèmes les plus difficiles que l'on peut rencontrer dans cette classe.

Pour les classes inférieures à NP dans la hiérarchie polynomiale, on utilise des notions plus fortes de réduction, comme la réduction en espace logarithmique. Le problème d'évaluation d'un circuit booléen est un exemple typique de problème P-complet : c'est un problème qui est résolvable en temps polynomial par une machine de Turing déterministe et qui permet la résolution de tout problème dans P par le biais de la réduction en espace logarithmique. Une réduction en espace logarithmique se définit comme une réduction qui est calculable sur une machine de Turing à trois rubans : un ruban d'entrée (accessible en lecture seulement) qui contient l'entrée, un ruban de travail (accessible en lecture et en écriture) qui est de taille logarithmique en la taille de l'entrée et permet de faire des calculs, et un ruban de sortie (accessible en écriture seulement) qui permet de rédiger la sortie.

1.8.5 Problème fonctionnel

Un problème fonctionnel est un problème qui, plutôt que d'associer une solution binaire à ses instances, associe une solution complexe. Un tel problème peut être vu comme une fonction de l'ensemble de ses instances à l'ensemble de ses sorties, à l'exception qu'une telle instance peut être négative et ne pas avoir de solution définie. Un tel problème se définit comme suit :

▶ Problème de factorisation d'un entier
Entrée: Un entier naturel n.
Sortie: La décomposition de n en facteurs premiers.

Les problèmes fonctionnels disposent de leur propres classes, qui sont la traduction naturelle des classes de décision. Les classes FP et FNP sont les classes dont les problèmes fonctionnels sont résolus par une machine de Turing respectivement déterministe et non déterministe en temps polynomial. La classe FcoNP regroupe les problèmes fonctionnels dont les instances sont prouvées ne pas avoir de solution en temps polynomial par une machine de Turing non déterministe.

La notion de réduction polynomiale est difficile à traduire dans le contexte des problèmes fonctionnels. Cela vient du fait que certains problèmes que nous voudrions définir comme FP-difficiles ne disposent d'aucune instance négative. Cette difficulté technique empêche la définition d'un ensemble de problèmes complets pour une classe de problèmes fonctionnels. Dans ce manuscrit, nous prouvons la difficulté d'un problème fonctionnel par réduction vers un problème de décision difficile.

Pour $f : A \to B$ une fonction, le problème fonctionnel qui correspond à f est le problème qui admet ses instances dans A, et qui pour toute instance $a \in A$ attend la réponse $f(a)$. Si ce problème est dans FP, alors la fonction f est dite calculable en temps polynomial. Dans le cas où ce problème peut être résolu par une machine de

Turing limitée à un espace de travail logarithmique en la taille de l'entrée, f est dite calculable en espace logarithmique.

1.8.6 Problème fonctionnel avec promesse

Les problèmes de complexité avec promesse permettent une sélection des instances du problème considéré, dont le coût calculatoire ne repose pas sur la résolution de l'instance. Une exploration plus approfondie de cette notion peut être trouvée dans (GOLDREICH 2006). Notre notation de ces problèmes est la suivante :

▶ Coloriage d'un graphe planaire

Entrée: Un graphe G non orienté, et k un entier naturel.

Promesse: Le graphe G est planaire.

Sortie: Une fonction f qui, à tout sommet de G, associe un entier naturel inférieur à k telle que pour deux sommets s, s', $(s, s') \in A \iff f(s) \neq f(s')$.

Pour résoudre l'ensemble des instances de ce problème, nous n'avons besoin que de considérer les instances pour lesquelles la promesse est vérifiée. Ainsi, tout algorithme résolvant ce problème a la possibilité d'adopter un comportement arbitraire sur les instances qui ne respectent pas la promesse. Il n'a donc jamais besoin de vérifier que l'entrée concernée encode un graphe planaire valide, puisque cela est garanti par la promesse.

2 Simulation : contexte et généralisations

La simulation est un concept singulier dans le calcul naturel. Bien que son intuition, la capacité de la reproduction du calcul, semble universelle, les définitions dont nous disposons sont toujours propres aux modèles qui les concernent. Ce schisme vient certainement du fait que nous n'avons jamais de définition unifiée de ce qu'est un calcul ; nous sommes réduits dans la pratique à ne définir que des exemples locaux de son apparition. En est-il de même avec la simulation, avec qui le calcul semble hautement apparenté ?

Nous ne sommes pas les premiers à nous poser la question (Boas 2014). Il est désirable d'obtenir une telle définition générale si elle existe, simplement car l'unification d'une partie des méthodes d'un large domaine de recherche est à la clé. Devant une question aussi ambitieuse, nous adoptons la démarche de décrire les intuitions qui semblent être clés dans les définitions de simulation dont nous disposons dans certains modèles de calcul comme les automates cellulaires, puis de poursuivre la généralisation de ces intuitions vers, nous l'espérons, une définition applicable dans un cadre plus général.

2.1 Une définition informelle

La notion de simulation est fortement liée à la notion de l'universalité, dont il existe des variantes. La plus classique d'entre elles, l'universalité Turing, prend racine dans la thèse de Church-Turing, des deux mathématiciens qui, dans les années 1930, déclarent indépendamment avoir défini des modèles de calcul qui ont la faculté de reproduire tout calcul qu'un humain peut opérer si on lui procure une quantité infinie de feuilles, de crayon et de temps. Bien que cette thèse n'ait jamais été prouvée (car trop informelle), elle est communément admise et n'a jamais été contredite depuis 90 ans. Les machines de Turing sont parmi les exemples les plus connus de modèles dit Turing universels.

Une fois admise l'universalité Turing d'un modèle, il est possible d'étendre cette universalité à des modèles parfois de natures complètement différentes. Pour ce faire, on utilise la notion de simulation. La simulation est une relation entre modèles de calcul qui s'applique entre un modèle simulateur et un modèle simulé. En pratique, tout calcul opérable par la machine simulée peut être reconstruit comme un calcul opérable par la machine simulatrice, de façon à ce que le calcul de la machine simu-

latrice permette de prédire tout aspect du calcul simulé. Une telle relation se devra d'être réflexive et transitive : une machine doit être capable de reproduire son propre calcul, ce qui est trivial ; si une machine en simule une seconde qui en simule une troisième, alors elle simule la troisième. Ces deux propriétés font de la simulation un pré-ordre.

Une autre variante de l'universalité est nommée universalité intrinsèque, et ne fait sens que lorsque l'on examine la capacité d'un modèle à simuler tout autre modèle inclus dans la même famille. Cela est généralement dit d'un modèle spécifiquement capable d'une grande souplesse calculatoire. C'est le cas de certaines machines de Turing universelles, qui sont conçues pour accepter et reproduire le code décrivant n'importe quelle autre machine : une telle machine est intrinsèquement universelle dans la famille des machines de Turing. L'universalité intrinsèque ne découle cependant pas toujours de l'universalité Turing. Pour être intrinsèquement universel, un modèle devra souvent opérer son calcul en exploitant les propriétés intrinsèques de la famille de modèles utilisés. Les conditions spécifiques de ces propriétés changent avec la famille considérée, mais en général impliquent une simulation plus efficace.

Afin de se construire une intuition du fonctionnement de la simulation, examinons des définitions tirées de la littérature.

2.1.1 Simulation d'automates cellulaires

Les automates cellulaires sont une famille de modèles centrale dans le calcul naturel. Une présentation sommaire d'une fraction emblématique de ces modèles, les automates cellulaires élémentaires, est faite en section 1.7.

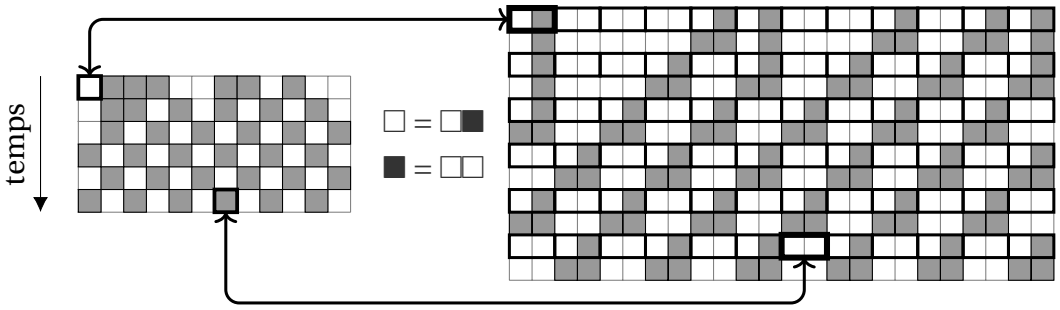

FIGURE 2.1 – Simulation d'une exécution de la règle 184 (gauche) par la règle 134 (droite). L'exécution de la règle 184 est encodée dans l'exécution de la règle 134. En ne considérant qu'une configuration sur deux, chaque paire de cellule (encadrées en noir sur la droite) encode une cellule de l'automate 184 de la façon suivante : l'état □□ encode la valeur ■ et l'état □■ encode la valeur □.

Lorsque l'on opère une simulation d'un automate cellulaire par un autre automate cellulaire, la méthode employée repose généralement sur l'encodage des valeurs des

cellules de l'automate simulé par des regroupements de cellules dans l'automate simulateur. La figure 2.1 détaille l'exemple de la simulation de la règle 184 par la règle 134 : ici, chaque automate de la règle 184 est remplacé par 2 cellules dans l'automate 134. On note également qu'une mise à jour supplémentaire est nécessaire entre chaque étape de la simulation. En d'autres termes, cette simulation opère par la mise à l'échelle dans l'espace et le temps de la règle 134. En choisissant une configuration initiale donnée par le bon encodage, la règle 134 se trouve reproduire "tous les calculs" de la règle 184.

Beaucoup d'autres simulations ont été développées dans la littérature des automates cellulaires, ce qui a permis l'identification de nombreux modèles intrinsèquement universels ou Turing universels (OLLINGER 2012 ; BECKER, MALDONADO, OLLINGER et al. 2018 ; COOK 2004 ; MARTIEL 2015). Un grand travail a été placé dans la généralisation des notions de simulation et d'universalité chez les automates cellulaires (OLLINGER 2002 ; THEYSSIER 2005), avec par exemple la généralisation de l'ensemble des méthodes reposant sur le groupage des cellules dans le simulateur par la notion de Bulking (DELORME, MAZOYER, OLLINGER et al. 2011a ; DELORME, MAZOYER, OLLINGER et al. 2011b), qui emploie des outils algébriques pour généraliser le plus fondamentalement possible la notion de mise à l'échelle de l'espace et du temps. Cette notion très générale de regroupement est ensuite utilisée pour définir plusieurs notions de simulation entre automates cellulaires.

La notion de Bulking permet une définition très générale de la simulation intrinsèque entre automates cellulaires ; c'est-à-dire toute simulation qui exploite les propriétés de localité et d'uniformité propres à ces modèles : leur nature uniforme et régulière invite à décrire une simulation efficace d'un automate par un autre comme une remise à l'échelle.

Cette définition est considérée satisfaisante pour englober les notions de simulation intrinsèque qui concernent les automates cellulaires. Qu'en est-il dans un cadre plus général ? Il existe par exemple des généralisations des automates cellulaires qui brisent la régularité de leur réseau ou l'uniformité de leur mise à jour dans l'espace ou le temps, comme les réseaux d'automates. Dans ces modèles, qui ne sont pas uniformes dans l'espace et le temps, le Bulking n'est pas applicable. De fait, l'intuition à la base de la notion de la simulation intrinsèque change radicalement dès que l'on change la famille de modèles que l'on considère.

2.1.2 Simulation de réseaux d'automates

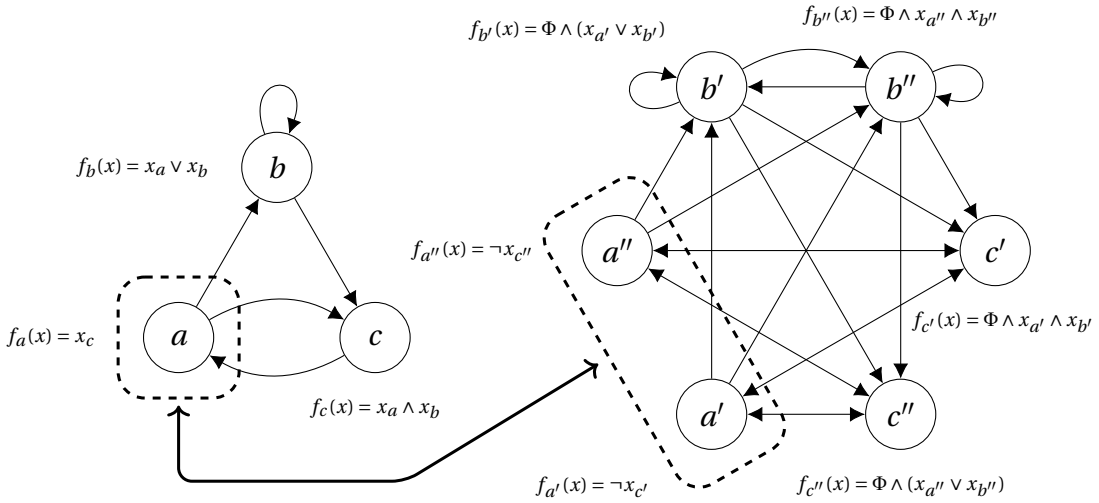

FIGURE 2.2 – Illustration de la simulation d'un réseau d'automates booléens de taille 3 (gauche) par un réseau d'automates booléens de taille 6 localement monotone (droite), dans lequel $\Phi = (x_{a'} \vee x_{a''}) \wedge (x_{b'} \vee x_{b''})$. Cette simulation est une application du corollaire 3 donné en section 5.3.3. L'état de chaque automate à gauche est encodé par les états de deux automates à droite ; par exemple, l'état de a est encodé par les états de a' et a''.

Les réseaux d'automates sont une variation des automates cellulaires qui ne sont pas définis sur un espace régulier. Si un automate cellulaire est composé de cellules, un réseau d'automates est composé d'éléments appelés automates. Les automates d'un réseau d'automates ne sont pas nécessairement mis à jour de façon homogène, ni même de façon synchrone. Un développement approfondi de ces modèles sera proposé au chapitre 4.

Puisque ces modèles ne garantissent pas la régularité de leur réseau ni celle de leur mise à jour, la réalisation d'une remise à l'échelle ne sera pas suffisante. À vrai dire, leur définition fait de chaque réseau d'automates un objet au graphe d'interaction particulier. Cependant, la plupart des simulations entre réseaux d'automates suit une logique similaire aux simulations entre automates cellulaires : chaque automate du réseau simulé sera représenté par un ou plusieurs automates dans le réseau simulateur. L'état de ce groupe d'automates encode alors l'état de l'automate simulé. Une illustration d'une telle simulation est représentée en figure 2.7, dans laquelle un réseau de taille 3 est simulé par un réseau de taille 6, tel que le comportement de chaque automate du réseau simulé est reproduit par une paire d'automates dans le simulateur. Cette méthodologie semble très similaire à celle développée dans l'exemple de la figure 2.1 : dans les deux cas, chaque automate est simulé localement par une petite structure. Une généralisation limitée de cette notion a été proposée par (PERROT, PERROTIN et

SENÉ 2018) et est également exprimée en section 5.3.3.

La simulation de tout réseau d'automates booléens par un réseau d'automates booléens localement monotone (PERROT, PERROTIN et SENÉ 2018) est un résultat qui permet l'établissement de l'universalité intrinsèque de ces derniers. La simulation de réseaux exécutés en parallèle par des réseaux exécutés par blocs-séquentiels (BRIDOUX, GUILLON, PERROT et al. 2017) en est un autre exemple. Une étude détaillée de la simulation dans les réseaux d'automates est faite par les travaux (BRIDOUX, CASTILLO-RAMIREZ et GADOULEAU 2020 ; BRIDOUX 2019 ; BRIDOUX, GADOULEAU et THEYSSIER 2020). Les notions liées aux réseaux d'automates sont développées en détail dans le chapitre 4.

2.2 Simulation par forme

La simulation intrinsèque semble en pratique souvent reposer sur une remise à l'échelle des modèles considérés : un élément simple est alors simulé par un ensemble d'éléments qui, mis ensemble, forment une structure au comportement recherché. Guidés par cette observation, nous développons une façon de percevoir les modèles de calcul qui permet la définition d'une notion de simulation, nommée simulation par forme. Son approche est très simple : les modèles de calcul sont perçus comme des collections de diagrammes espace-temps. Chacun de ces diagrammes assigne un état à chaque élément de l'espace et du temps. La simulation par forme repose sur l'encodage d'un tel diagramme par l'assemblage d'un diagramme simulateur obtenu par le remplacement de chaque élément atomique du diagramme simulé par une "forme".

2.2.1 Modèles de forme et sous-espaces fonctionnels

Comme décrit plus haut, les modèles de calcul sur lesquels la simulation par forme se définit sont considérés comme des collections d'exécutions. Dans cette section, l'ensemble X sera en général associé avec l'ensemble des positions dans l'espace et le temps d'un modèle de calcul. Dans beaucoup d'exemples, X est le produit cartésien d'un ensemble spatial P et d'un ensemble temporel T.

Exemple 17. *Considérons un automate cellulaire élémentaire. L'ensemble d'espace-temps X qui correspond à ce modèle est le produit cartésien $X = P \times T$, où $P = \mathbb{Z}$ correspond à l'espace d'une configuration infinie, et $T = \mathbb{N}$ correspond au temps, qui est infini avec une configuration initiale* 0. *Chaque paire $(p, t) \in X$ correspond au moment t de la cellule à la position p.*

L'ensemble S sera quant à lui associé à l'ensemble des états que chaque position dans l'espace-temps peut admettre.

Exemple 18. *Considérons un modèle d'automate cellulaire élémentaire. Ce modèle est booléen, et son espace d'états est $S = \mathbb{B}$.*

Dans le formalisme de la simulation par forme, une *exécution* d'un modèle est décrite par une fonction $e : X \to S$. Ici l'exécution e associe à tout élément dans l'espace-temps un état dans S. Un *modèle de forme* est défini par trois éléments : l'ensemble X, l'ensemble S, et l'ensemble des exécutions qui sont admises par le modèle. Ce dernier ensemble est généralement noté E, et est par définition un sous-ensemble de l'espace des fonctions $X \to S$. Nous appellons parfois E un *sous-espace fonctionnel*. Le modèle est décrit par le triplet $M = (X, S, E)$.

Exemple 19. *Soit $X = \mathbb{Z} \times \mathbb{N}$ et $S = \mathbb{B}$. Soit $f_k : \mathbb{B}^3 \to \mathbb{B}$ la fonction qui définit l'automate cellulaire élémentaire numéro k (voir définition en section 1.7). Le modèle de forme de cet automate est défini par le sous-espace fonctionnel E_k qui contient exactement les exécutions e telles que, pour tout x dans X et t dans T,*

$$e(x, t+1) = f_k(e(x-1, t), e(x, t), e(x+1, t)).$$

On note $M_k = (X, S, E_k)$ le modèle correspondant.

La définition d'un modèle de forme sélectionne les exécutions qui lui correspondent.

2.2.2 Formes

Les formes sont toutes les possibles macro-structures qui peuvent être construites dans l'exécution d'un modèle. Dans des exemples concrets de simulation, le comportement de ces macro-structures aura un comportement équivalent, dans le bon contexte, aux éléments atomiques du modèle simulé. Dans le cadre de la simulation par forme, nous proposons la définition de ces formes comme étant n'importe quelle disposition fixée d'éléments dans le modèle considéré. En ces termes, une forme est un ensemble de paires dans $X \times S$.

Définition 1 (Forme)**.** *Soit $M = (X, S, E)$ un modèle. Une* forme *h de M est un sous-ensemble de $X \times S$.*

Exemple 20. *Soit M_{134} le modèle d'automate cellulaire élémentaire défini dans l'exemple 19. Pour tout $b \in \mathbb{B}$, $x \in \mathbb{Z}$ et $t \in \mathbb{N}$, on définit la forme $h_{x,t,b}$ telle que*

$$h_{x,t,0} = \{((2x, 2t), 0), ((2x+1, 2t), 1)\}$$
$$h_{x,t,1} = \{((2x, 2t), 0), ((2x+1, 2t), 0)\}$$

Une forme peut être définie comme n'importe lequel de ces sous-ensembles. Ainsi, on pourra observer une forme vide, ou bien une forme qui définit deux valeurs à une seule position de l'espace et du temps. On pourra même considérer l'exemple extrême d'une forme qui définit toutes les paires possibles. Toutes ces choses sont des formes. Ainsi, l'ensemble des formes de M est simplement l'ensemble $\wp(X \times S)$.

Une forme est dite observée dans une exécution si et seulement si l'ensemble des paires (x, s) qui la composent se retrouvent dans l'exécution en question.

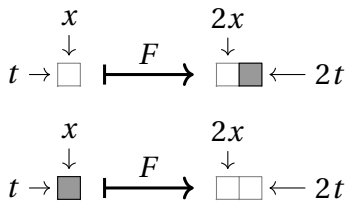

FIGURE 2.3 – Illustration de l'encodage opéré par la fonction de forme F définie dans l'exemple 22. Ici, les cases blanches représentent la valeur 0 et les cases noires représentent la valeur 1, similairement à la façon dont nous représentons les exécutions d'un automate cellulaire élémentaire.

Définition 2 (Observation de forme)**.** *Soit e une exécution et h une forme. La forme h est* observée *dans e lorsque $h \subseteq e$.*

Exemple 21. *On considère M_{134} le modèle défini dans l'exemple 19 et $h_{x,t,b}$ la famille de formes définie dans l'exemple 20. Pour toute exécution e de M_{134}, On remarque que pour tout $x \in \mathbb{Z}$, $t \in \mathbb{N}$, et tout triplet $(b_1, b_2, b_3) \in \mathbb{B}^3$,*

$$h_{x-1,t,b_1} \cup h_{x,t,b_2} \cup h_{x+1,t,b_3} \subseteq e \implies h_{x,t+1,f_{184}(b_1,b_2,b_3)} \subseteq e.$$

2.2.3 Fonction de forme

Pour décrire une simulation par un modèle $M = (X, S, E)$ d'un modèle $M' = (X', S', E')$, nous devons décrire la traduction des éléments atomiques de M' en termes de formes de M. Nous utilisons pour cela une fonction de forme.

Définition 3 (Fonction de forme)**.** *Une* fonction de forme F *de M' dans M est une fonction définie avec domaine $(X' \times S')$ et co-domaine $\wp(X \times S)$.*

Exemple 22. *Soit M_{134} notre modèle simulateur et M_{184} notre modèle simulé, définis dans l'exemple 19. Soit $h_{t,x,b}$ l'ensemble de formes défini dans l'exemple 20. Nous définissons F la fonction de forme qui vérifie*

$$F((x,t),b) = h_{t,x,b}.$$

Le calcul de cette fonction est représenté dans la figure 2.3.

La fonction de forme est fondamentalement locale ; elle permet la traduction d'un seul élément atomique du modèle simulé dans le modèle simulateur. Cette fonction peut également se comprendre comme une fonction d'encodage. Nous utilisons la fonction de forme dans les deux sens ; si elle permet d'obtenir l'encodage d'un élément simple, nous l'utilisons également pour déduire d'une exécution du modèle simulateur l'ensemble des éléments atomiques encodés dans cette exécution. Ce procédé est nommé interprétation de forme.

2.2.4 Interprétation de forme

Si la fonction de forme F est une fonction locale d'encodage, l'interprétation de forme repose sur une fonction f_F qui identifie l'ensemble des éléments atomiques encodés dans une exécution donnée du simulateur. Cette fonction de décodage est définie comme suit pour h toute forme de M :

$$f_F(h) = \{(x', s') \mid F(x', s') \subseteq h\}.$$

En général, cette fonction est définie sur $f_F : \wp(X \times S) \to \wp(X' \times S')$, ce qui signifie qu'elle admet l'ensemble des formes du simulateur en domaine et l'ensemble des formes du simulé en co-domaine.

Intuitivement, f_F effectue une collection de l'ensemble des éléments atomiques (x', s') dont la forme $F(x', s')$ est observée dans h. Ainsi, lorsque l'on prend une exécution e de M et qu'on l'interprète comme un ensemble de paires du domaine et co-domaine de e, l'évaluation de $f_F(e)$ retourne une forme composée de l'union des paires qui produisent par F les formes observées dans e. Dans certains cas, la forme retournée par $f_F(e)$ est complète dans le sens qu'il existe exactement une valeur $s' \in S'$ telle que $(x', s') \in f_F(e)$, pour tout $x' \in X'$. Dans ce contexte, il est valide de considérer $f_F(e)$ comme une exécution e' de M', auquel cas e' est l'interprétation de e d'après F.

2.2.5 Simulation par forme

Pour qu'un modèle M simule un modèle M', le choix de la fonction de forme F doit respecter un certain nombre de propriétés. D'abord, toute exécution de e' doit avoir une exécution de e dont elle est l'interprétation d'après F. Ensuite, il ne doit pas exister d'exécution e telle que l'interprétation de e est une fonction totale, qui n'est pas une exécution. Ce dernier point est important pour que la simulation comporte précisément l'information contenue dans le modèle M.

Définition 4 (Simulation par forme)**.** *Le modèle M* simule *le modèle M', noté $M' \preccurlyeq M$, si et seulement s'il existe une fonction de forme F de M' dans M telle que*

1. *Pour toute e' dans E', il existe e dans E telle que $f_F(e) = e'$.*
2. *Pour toute e dans E, si $f_F(e) \in X' \to S'$ (c'est à dire si $f_F(e)$ est fonctionnelle et totale), alors $f_F(e) \in E'$.*

La première condition est simple : elle requiert que le décodage donné par F permette de retrouver l'ensemble des exécutions de M' depuis M. La seconde condition est plus complexe : elle stipule que l'interprétation de toute exécution de M doit résulter en une forme qui doit soit être une exécution de M', soit être incomplète ou contradictoire. Dans le cas où cette condition n'est pas vérifiée, c'est qu'il existe une exécution e telle que $f_F(e)$ est une fonction $e' : X' \to S'$ bien définie qui n'est pas une exécution admise par M', ce qui est problématique car cela démontre que la simulation de M' par M "rajoute" de l'information au calcul de M' sans en respecter

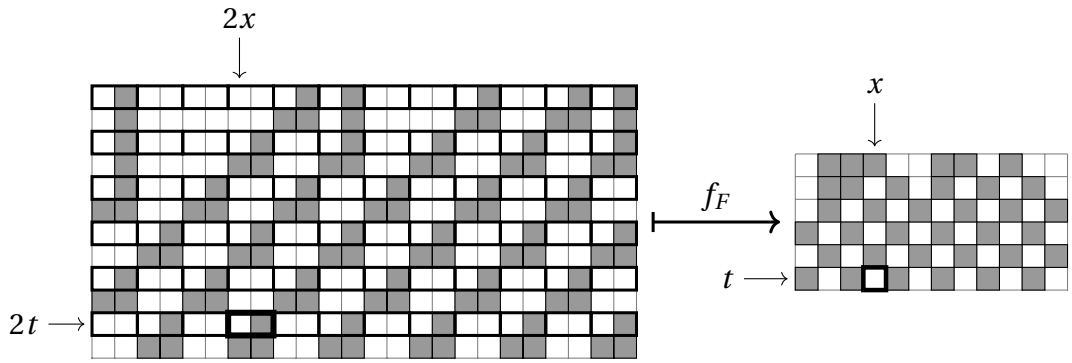

FIGURE 2.4 – Interprétation d'une exécution du modèle M_{134} (gauche) en une forme du modèle M_{184} (droite) par le biais de la fonction d'interprétation f_F dérivée de la fonction de forme F définie dans l'exemple 22. La forme obtenue dans cette illustration est un fragment valide d'une exécution du modèle M_{184}.

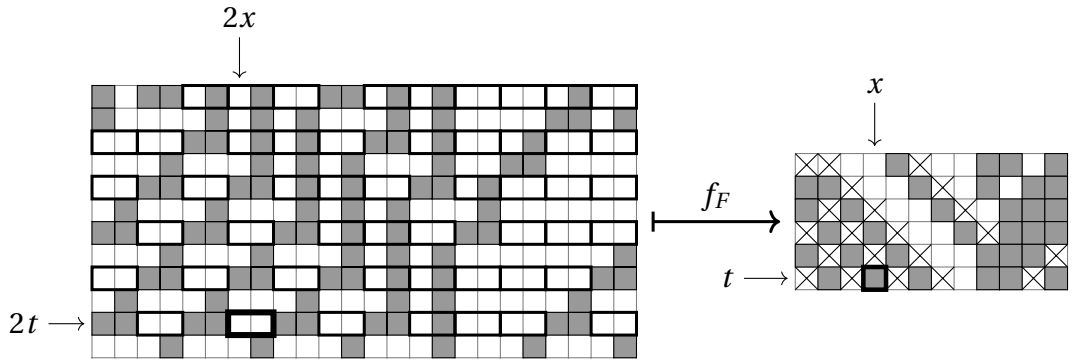

FIGURE 2.5 – Interprétation d'une exécution du modèle M_{134} (gauche) en une forme du modèle M_{184} (droite) par le biais de la fonction d'interprétation f_F dérivée de la fonction de forme F définie dans l'exemple 22. La forme obtenue dans cette illustration est incomplète : chaque croix représente un point dans l'espace et le temps qui n'a pas de valeur définie par f_F.

les règles. Cette seconde condition permet de vérifier que toute interprétation d'une exécution de M est soit une exécution admise par M', soit une forme incomplète ou contradictoire, et donc non considérée dans le contexte de la simulation.

Exemple 23. *Soient M_{134} et M_{184} les modèles de forme et F la fonction de forme définis dans l'exemple 22. Nous reportons l'observation de l'exemple 21 que pour tout $x \in \mathbb{Z}$, $t \in \mathbb{N}$, et tout triplet $(b_1, b_2, b_3) \in \mathbb{B}^3$,*

$$h_{x-1,t,b_1} \cup h_{x,t,b_2} \cup h_{x+1,t,b_3} \subseteq e \implies h_{x,t+1,f_{184}(b_1,b_2,b_3)} \subseteq e.$$

Par cette observation, nous établissons que $M_{184} \preccurlyeq M_{134}$. En effet, pour toute configuration initiale dans M_{184}, il existe une exécution de M_{134} dont la configuration intiale l'interprète par la fonction de forme F. L'application de l'observation permet de déduire que le reste de l'exécution de M_{184} est également proprement interprété.

On observe également qu'aucune exécution de M_{134} ne peut être interprétée de façon contradictoire; c'est-à-dire que les formes $F((x,t),0)$ et $F((x,t),1)$ ne peuvent jamais être définies dans la même exécution. Cela signifie que l'interprétation de toute exécution de M_{134} est soit incomplète, soit complète; dans le second cas, notre précédente observation implique qu'il s'agit d'une exécution valide de M_{184}, et la seconde condition de simulation est donc vérifiée.

Cet exemple est une application de la simulation par forme sur un exemple de simulation classique illustré en figure 2.1.

Une application de la fonction d'interprétation f_F dérivée de la fonction de forme F définie dans l'exemple 22 est illustrée en figures 2.4 et 2.5. La figure 2.4 représente une exécution du modèle M_{134} qui est interprétée par f_F en une exécution valide du modèle M_{184}. La figure 2.5 représente une exécution e du modèle M_{134} dont l'interprétation par f_F donne une information incomplète, ce qui signifie que l'exécution e ne simule aucune exécution du modèle M_{184}.

2.2.6 Critiques

Intuitivement, la simulation est comprise comme une relation qui démontre qu'un modèle est au moins aussi expressif qu'un autre : puisque tout modèle est au moins aussi expressif que lui-même, la réflexivité est une propriété qui tombe sous le sens. De façon similaire, la transitivité affirme que si un premier modèle est démontré aussi expressif qu'un second modèle, qui est démontré aussi expressif qu'un troisième modèle, alors le premier modèle est aussi expressif que le troisième modèle. Ces deux propriétés mises ensemble font de la simulation un pré-ordre. Sont-elles vérifiées en pratique dans le contexte de la simulation par forme ?

Propriété 1. *Tout modèle de forme M vérifie $M \preccurlyeq M$.*

Démonstration. La fonction de forme $F(x,s) = \{(x,s)\}$ permet pour tout modèle de forme de définir la fonction d'interprétation f_F suivante :

$$\begin{aligned} f_F(e) &= \{(x,s) | F(x,s) \subseteq e\} \\ &= \{(x,s) | \{(x,s)\} \subseteq e\} = e \end{aligned}$$

ce qui est l'identité sur les formes de M et qui vérifie trivialement toutes les conditions de la simulation par forme. □

Si la réflexivité est facile à vérifier, il n'en est pas de même pour la transitivité. Pour trois modèles M, M', M'' tels que $M' \preccurlyeq M$ et $M'' \preccurlyeq M'$, et pour F et F' les fonctions de forme de ces simulations respectives, une proposition naturelle de démonstration

que $M'' \preccurlyeq M$ repose sur la création d'une fonction de forme F'' définie comme une forme de composition de F et F' comme suit :

$$F''(x'', s'') = \{F(x', s') | (x', s') \in F'(x'', s'')\}.$$

Cette fonction applique simplement la fonction de forme F à tous les éléments de la forme retournée par la fonction de forme F'. Notons cette forme de composition la *composition naturelle* de F et F'.

Propriété 2. *Il existe M, M', M'' des modèles de forme et F, F' des fonctions de forme tels que F vérifie $M' \preccurlyeq M$ et F' vérifie $M'' \preccurlyeq M'$, mais tels que la composition naturelle de F et F' ne vérifie pas $M'' \preccurlyeq M$.*

Démonstration. Nous établissons un contre-exemple pour démontrer le résultat. On considère $M = M_{134}$, $M' = M_{184}$ et F tels que définis dans l'exemple 22. On construit $M'' = (X'', S'', E'')$ le modèle de forme qui vérifie $X'' = \{a, b\}$, $S'' = \{0, 1, 2, 3\}$ et $e'' \in E''$ si et seulement si $e''(a) \geq 2$ et $e''(a) \geq e''(b)$. Pour démontrer que $M'' \preccurlyeq M_{184}$, On construit F' la fonction de forme qui vérifie

$$\begin{aligned}
F'(a,2) &= \{((1,0),0)\} \\
F'(a,3) &= \{((1,0),1)\} \\
F'(b,0) &= \{((0,1),0),((1,1),0)\} \\
F'(b,1) &= \{((0,1),0),((1,1),1)\} \\
F'(b,2) &= \{((0,1),1),((1,1),0)\} \\
F'(b,3) &= \{((0,1),1),((1,1),1)\}.
\end{aligned}$$

Pour vérifier cette simulation, observons que, pour toute exécution e' de M_{184}, si $F'(a,2) \in e'$, alors la case 1 de la configuration initiale de e' est à 0. En théorie, si la valeur de a est 2, alors la valeur de b peut être tout sauf 3. Cela est vérifié en pratique puisque l'observation de $F'(a,2) \in e'$ implique par la règle locale de l'automate cellulaire élémentaire 184 que les cases 0 et 1 de la seconde configuration de e' peuvent admettre toutes valeurs sauf 1 et 1. On en déduit que si $F'(a,2) \in e'$, alors $F'(b,k) \in e'$ est possible si et seulement si $k \leq 2 = a$.

Lorsque l'on suppose $F'(a,3) \in e'$, les cases 0 et 1 de la configuration suivante ne sont pas restreintes, et b peut être observé avec n'importe quelle valeur de 0 à 3, ce qui vérifie $M'' \preccurlyeq M_{184}$.

On définit maintenant F'' comme la composition naturelle de F et F' :

$$F''(x'', s'') = \{F(x', s') | (x', s') \in F'(x'', s'')\}.$$

Soit e_I l'exécution de M_{134} qui possède pour configuration initiale la répétition du motif 1, 0, 0. Cette exécution, interprétée par f_F, donne une information incomplète et ne change donc pas la nature de la simulation $M_{184} \preccurlyeq M_{134}$. Cependant, son interprétation par $f_{F''}$ donne une forme de M'' qui est une fonction bien définie, vérifiant

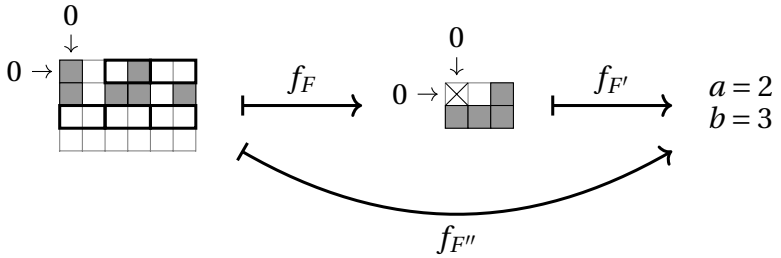

FIGURE 2.6 – Illustration du contre-exemple développé dans la preuve de la propriété 2, comprenant une exécution du modèle M_{134} (gauche), une forme du modèle M_{184} (centre) et une forme du modèle M'' (droite). La croix dans la forme du centre indique que l'exécution de gauche est interprétée par f_F de façon incomplète. De plus, l'interprétation de cette forme incomplète par $f_{F'}$ donne une exécution invalide de M'', avec $a < b$. On en déduit que l'interprétation de l'exécution de gauche par $f_{F''}$ donne une exécution invalide du modèle M'', ce qui implique que F'' ne définit pas une simulation.

$f_{F''}(a) = 2$ et $f_{F''}(b) = 3$, ce qui n'est pas une exécution valide de M'' (voir figure 2.6). La fonction de forme F'', définie comme la composition naturelle de F et F', ne permet donc pas de vérifier $M'' \preccurlyeq M_{134}$.

□

Si la composition de F et F' ne garantit pas une simulation, la question de la transitivité de la simulation par forme reste ouverte. En effet, il reste à décider l'existence d'une autre fonction de forme F'', qui ne serait pas la composition naturelle de F et de F', et qui permet une simulation. Il ne parait pas facile de trouver une méthode simple pour générer cette fonction F'' ; et s'il n'en existe aucune, la mise en pratique de la simulation par forme devient difficile.

Question ouverte 1. *La simulation par forme est-elle transitive ?*

D'une façon plus générale, ce formalisme tente de ne pas considérer l'axe du temps comme une dimension identifiée de ses modèles. Cette particularité rend la représentation de concepts comme la causalité, le déterminisme, ou même des notions plus triviales comme la configuration initiale ou finale d'un calcul impossible dans le cas le plus général.

De plus, ce formalisme se limite à des interprétations fondée sur la composition simple de formes prédéfinies dans l'espace et le temps. Cela peut avoir pour conséquence d'empêcher l'inclusion de certains résultats de simulation dès lors qu'ils concernent des modèles dans lesquels la position dans l'espace-temps de certaines macro-structures ne sont pas garanties avec une précision absolue. Parmi les modèles concernés, nous pouvons citer les modèles de particules, dans lesquels le calcul prend la forme d'interaction de particules mouvantes. Dans la plupart de ces modèles, les

détails précis du calcul considéré peuvent décaler une particule donnée selon la situation, sans entraver la simulation en cours (COOK 2004). Pourtant, la simulation par forme sera incapable d'interpréter cette particule de façon uniforme si sa position exacte n'est pas garantie.

2.3 Préordre de simulation

Parmi les nombreuses préoccupations du domaine de l'informatique théorique réside la motivation de généraliser et comprendre la notion de langage de programmation. Ce domaine large englobe de très nombreux modèles tels que les automates finis (SALOMAA 2014) ou le lambda-calcul (HINDLEY et SELDIN 2008). Lorsqu'il devient nécessaire de comparer le calcul de deux machines, ce domaine utilise la notion très générale de bisimulation ; qui considère que deux modèles sont bisimilaires si toute opération faite par l'un peut être reproduite par l'autre (MILNER 1971 ; SANGIORGI 2011).

Dans cette section, nous explorons une définition de bisimulation et proposons des modifications qui permettent de l'employer dans le contexte des modèles de calcul naturel.

Le travail présenté dans cette section est le fruit d'une recherche en collaboration avec Guilhem Gamard, Pierre Guillon, Kévin Perrot, Enrico Porreca, Martín Ríos Wilson, Sylvain Sené, Guillaume Theyssier et d'autres.

2.3.1 Définitions

La bisimulation, et la relation de préordre de simulation qui la compose, sont des concepts qui permettent la comparaison du calcul des systèmes de transition d'états. Un système de transition d'états est un graphe orienté, dont les sommets et les arêtes sont étiquetés.

Définition 5 (Système de transition d'états)**.** *Pour S un ensemble d'états et Λ un ensemble d'étiquettes, un* système de transition d'états *est un triplet $(S, \Lambda, \to)$, avec $\to$ un sous-ensemble de $S \times \Lambda \times S$.*

Les sommets de ce graphe sont parfois appelés états ou processus, et les étiquettes des arêtes parfois appelées actions. Un triplet $(s, a, s') \in \to$ s'interprète en disant que depuis l'état s, l'action a permet d'évoluer dans l'état s', ce que l'on note $s \xrightarrow{a} s'$. Lorsque Λ est un singleton, on considère que le système n'est pas étiqueté, et on considère alors $\to$ comme une simple relation $\to \subseteq S \times S$.

Exemple 24. *Soit S l'ensemble des configurations $\mathbb{B}^{\mathbb{Z}}$, Λ un singleton et k un entier entre 0 et 255. On définit $\to_{ECA(k)}$ la relation qui vérifie $s \to s'$ si et seulement si s' s'obtient par mise à jour de l'automate cellulaire élémentaire numéro k sur la configuration s. Le triplet $(S = \mathbb{B}^{\mathbb{Z}}, \Lambda = \{a\}, \to_{ECA(k)})$ est le système de transition d'états qui correspond à l'automate cellulaire élémentaire numéro k.*

La bisimulation est une relation qui se définit entre deux états d'un unique système. Ces états sont dits bisimilaires si toute action opérable dans un de ces états peut être opérée dans l'autre de façon à ce que les nouveaux états obtenus soient également bisimilaires.

Définition 6 (Bisimulation)**.** *Pour* $(S, \Lambda, \rightarrow)$ *un système de transition d'états, une* bisimulation *est une relation* $\mathscr{R} \subseteq S \times S$ *qui vérifie que, dès que* $s \mathscr{R} r$*, de façon symétrique*

— *pour tout* s' *tel que* $s \xrightarrow{a} s'$*, il existe* r' *tel que* $r \xrightarrow{a} r'$ *et* $s' \mathscr{R} r'$

— *pour tout* r' *tel que* $r \xrightarrow{a} r'$*, il existe* s' *tel que* $s \xrightarrow{a} s'$ *et* $s' \mathscr{R} r'$*.*

La *bisimilarité*, notée $\sim$, est l'union de toutes les bisimulations. C'est-à-dire que $s \sim r$ implique qu'il existe une bisimulation $\mathscr{R}$ telle que $s \mathscr{R} r$.

Une façon de lire ces définitions est que deux états s et s' font partie d'une bisimulation si et seulement si toute action de l'un peut être reproduite par une action de l'autre. Cette définition s'intéresse en un sens à l'équivalence de deux modèles dynamiques non pas par leur calcul final, mais par l'équivalence et l'inter-reproductibilité de chacune de leurs étapes de calcul.

La bisimulation est naturellement définie comme la juxtaposition de deux relations : que le calcul de s puisse être reproduit par s', et vice-versa. La version de la bisimulation qui casse cette symétrie est appellée préordre de simulation, et sa définition est intuitivement dérivée de celle de la bisimulation.

Définition 7 (Préordre de simulation)**.** *Pour* $(S, \Lambda, \rightarrow)$ *un système de transition d'états, un* préordre de simulation *est une relation* $\mathscr{S} \subseteq S \times S$ *telle que* $s \mathscr{S} r$ *implique que pour tout* s' *tel que* $s \xrightarrow{a} s'$*, il existe* r' *tel que* $r \xrightarrow{a} r'$ *et* $s' \mathscr{S} r'$*.*

Ce formalisme permet de définir que le calcul à partir d'un état peut être reproduit en tout point à partir d'un autre. Cette notion commence à s'approcher d'une notion de simulation dans le contexte des modèles de calcul naturel. Cependant, dans la plupart des simulations de la littérature des modèles de calcul naturel, le calcul des modèles considérés n'a pas de correspondance étape de calcul par étape de calcul. On peut prendre pour exemple la simulation représentée en figure 2.1, dans laquelle deux étapes de la règle 134 sont nécessaires pour reproduire une étape de la règle 184. Tenter d'appliquer la définition de préordre de simulation sur ces modèles vus comme des systèmes de transition d'états ne permettra pas de représenter la majorité des simulations qui ont été développées dans le domaine.

Nous nous intéressons donc à une variation de ce préordre de simulation qui permet l'inclusion de plus d'une étape de calcul pour reproduire l'étape opérée par le modèle simulé. Cette variation est le préordre de simulation faible, défini comme suit.

Définition 8 (Préordre de simulation faible)**.** *Pour* $(S, \Lambda, \rightarrow)$ *un système de transition d'états et* $\tau \in \Lambda$ *une action dite silencieuse, un* préordre de simulation faible *est une relation* $\mathscr{S} \subseteq S \times S$ *telle que* $s \mathscr{S} r$ *implique*

— *Pour tout* s' *tel que* $s \xrightarrow{\tau} s'$*, il existe* r' *tel que* $r \xrightarrow{\tau^*} r'$ *et* $s' \mathscr{S} r'$*.*

— *Pour tout s' tel que $s \xrightarrow{a} s'$, il existe r' tel que $r \xrightarrow{\tau^* a \tau^*} r'$ et $s' \mathscr{S} r'$.*

Dans cette définition, la notation τ^* correspond à l'application de l'étoile de Kleene sur l'action τ qui décrit la répétition de cette opération un nombre fini de fois. Dans le contexte du préordre de simulation faible, l'action dite silencieuse τ correspond à des actions intermédiaires qui ne sont pas considérées comme de véritables actions observables du modèle concerné. Ainsi on pourra considérer la concaténation d'un nombre arbitraire d'action silencieuses τ^* comme une opération effectivement silencieuse, ce qui permet en pratique de concaténer plusieurs opérations en une seule et permettre de rendre la définition de simulations entre modèles plus souple.

2.3.2 Travail d'adaptation

La bisimulation et ses définitions dérivées sont fondées sur des notions propres à des domaines assez différents du domaine du calcul naturel. Cela implique certaines différences formelles qu'il faut observer lorsque l'on souhaite proposer une adaptation de ces définitions d'un domaine vers l'autre.

Une différence notable est la définition des étiquettes des transitions. Dans le contexte des modèles de calcul, différents modèles risquent d'avoir des actions radicalement différentes ; on citera par exemple l'ensemble des modes de mise à jour possibles pour un réseau d'automates, ou l'ensemble des choix d'une machine de Turing non déterministe. Il n'y a pas de raison de trouver une nomenclature universelle pour tous ces modèles bien distincts, à raison que le choix de cette nomenclature a une influence sur les simulations qui pourraient être alors définies. Ainsi, pour tenter de produire une définition pratique et générale, nous introduisons une fonction g qui permet la traduction de l'action simulée en une série d'actions du simulateur. Cela permet également d'éviter la convention de l'action silencieuse, qui ne fait que rarement sens dans le contexte des modèles de calcul naturel.

Il est également intéressant de noter que les définitions présentées dans la sous-section précédente s'appliquent uniquement sur les états d'un seul système. Cela permet une perspective générale, car la simulation entre deux modèles peut être comprise comme la simulation de paires d'états dans le système de transition d'états construit comme l'union des systèmes de chaque modèle séparé.

Enfin, comme nous ne nous intéressons ici qu'à la simulation d'un modèle par un autre, il nous semble plus simple de remplacer la relation $\mathscr{S}$ par une fonction partielle que nous appellons h. Cette fonction a pour domaine un sous-ensemble des états du modèle simulateur, et pour co-domaine les états du modèle simulé.

Définition 9 (Simulation)**.** *Pour $(S, \Lambda, \rightarrow)$ un modèle A et $(S', \Lambda', \rightarrow')$ un modèle B, le modèle A est* simulé *par le modèle B s'il existe deux fonctions $g : \Lambda \rightarrow \Lambda'^*$ et $h : S' \twoheadrightarrow S$ telles que h est surjective, et*

1. *pour tout $s, r \in S$, $a \in \Lambda$ et $s' \in dom(h)$, si $h(s') = s$ et $s \xrightarrow{a} r$, alors il existe $r' \in dom(h)$ tel que $h(r') = r$ et $s' \xrightarrow{g(a)}{}' r'$ (tout calcul est simulé)*

2. *pour tout $s \in S$, $a \in \Lambda$, et $s', r' \in dom(h)$, si $h(s') = s$ et $s' \xrightarrow{g(a)}{}' r'$ alors $s \xrightarrow{a} h(r')$ (tout calcul simulé est un calcul).*

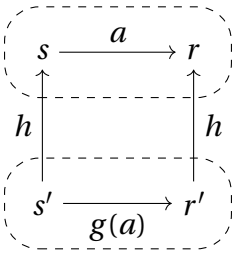

FIGURE 2.7 – Diagramme représentant les relations entre les états s, r d'un modèle simulé (en haut) et s', r' d'un modèle simulateur (en bas). La surjectivité de h implique que pour tout s, r, il existe s', r' tels que $h(s') = s$ et $h(r') = r$. La première clause de la définition 9 permet de vérifier que dès que l'on peut aller de s à r par l'action a, alors on peut aller de tout s' vers tout r' par la séquence d'actions $g(a)$. La seconde clause de la définition permet de vérifier que pour tout s', dès que l'on peut aller vers un r' par la séquence d'actions $g(s)$, alors on peut aller de s à r par l'action a.

Cette définition introduit deux fonctions, g et h, qui définissent la transformation qui permet la simulation d'un modèle par un autre. La fonction h est une fonction de décodage : pour s' une configuration du modèle simulateur, $h(s')$ calcule la configuration s du modèle simulé qui est encodée dans s'. Cette fonction est partielle, car toutes les configurations du simulateur n'ont pas vocation à encoder une configuration du modèle simulé. Cette fonction est surjective, car toute configuration simulée doit être encodée par au moins une configuration du simulateur. Elle n'est cependant pas nécessairement injective, car il peut exister plusieurs façons d'encoder une configuration donnée dans le simulateur. La fonction g, pour a toute action du simulé, calcule la séquence d'actions qui reproduit a dans le simulateur.

Nous proposons de créer des notions plus spécifiques de simulations en mesurant g et h en complexité.

Définition 10 (Simulation en espace logarithmique)**.** *On définit que A est simulé par B en espace logarithmique, noté $A \preccurlyeq_L B$, si A est simulé par B par le biais de fonctions g et h telles que*

- *g et h sont calculables en espace logarithmique*
- *pour tout état s de A, on peut calculer un état s' de B tel que $h(s') = s$ en espace logarithmique.*

Définition 11 (Simulation en temps polynomial)**.** *On définit que A est simulé par B en temps polynomial, noté $A \preccurlyeq_P B$, si A est simulé par B par le biais de fonctions g et h telles que*

— *g et h sont calculables en temps polynomial*
— *pour tout état s de A, on peut calculer un état s' de B tel que $h(s') = s$ en temps polynomial.*

Il est important de garantir que ces définitions restent des définitions de préordres, c'est-à-dire des relations binaires réflexives et transitives, ce que nous vérifions par les propriétés suivantes.

Propriété 3. *La simulation en espace logarithmique vérifie*
— *la réflexivité : tout modèle A vérifie $A \preccurlyeq_L A$;*
— *la transitivité : pour tous modèles A, B, C, si $A \preccurlyeq_L B$ et $B \preccurlyeq_L C$, alors $A \preccurlyeq_L C$.*

Démonstration. Soit A un modèle. En prenant h et g comme les identités sur S et Λ respectivement, nous vérifions que h est une bijection, et que $s \xrightarrow{a} r$ est équivalent à $h^{-1}(s) \xrightarrow{g(a)} h^{-1}(r)$. Ces fonctions, qui recopient simplement leur entrée sur leur sortie, sont bien calculables en espace logarithmique. Cela prouve que la simulation en espace logarithmique est réflexive.

Soient $A = (S, \Lambda, \rightarrow), B = (S', \Lambda', \rightarrow')$ et $C = (S'', \Lambda'', \rightarrow'')$ trois modèles. Par hypothèse, A est simulé en espace logarithmique par B par le biais des fonctions g et h, et B est simulé en espace logarithmique par C par le biais de fonctions g' et h'. Le reste de cette preuve consiste à prouver que nous pouvons simuler A par C en espace logarithmique par le biais de fonctions g'' et h''.

Nous construisons $h'' : S'' \nrightarrow S$ telle que $h''(s'') = (h \circ h')(s'')$, définie pour tout s'' qui vérifie $s'' \in \mathrm{dom}(h')$ et $h'(s'') \in \mathrm{dom}(h)$.

Nous construisons $g'' : \Lambda \rightarrow \Lambda''^*$ comme la composition naturelle de g et g', c'est à dire que pour tout $a \in \Lambda$, $g''(a)$ est la concaténation des séquences obtenues par l'application de g' sur chaque élément de la séquence $g(a)$, dans l'ordre.

Par hypothèse que h, h', g et g' sont calculables en espace logarithmique, les fonctions g'' et h'' sont également calculables en espace logarithmique.

Puisque par hypothèse pour tout s on peut construire s' en espace logarithmique tel que $h(s') = s$ et pour tout s' on peut construire s'' en espace logarithmique tel que $h'(s'') = s'$, on peut, pour tout s, construire s'' en espace logarithmique tel que $h''(s'') = s$. Enfin, h'' est surjective par construction.

Pour voir que ces fonctions définissent bien une simulation de A par C, considérons s, r dans S, a dans Λ et $s'' \in S''$ tels que $s \xrightarrow{a} r$, et $h''(s'') = s$.

Si $s \xrightarrow{a} r$, alors il existe $s', r' \in \mathrm{dom}(h)$ tels que $s' \xrightarrow{g(a)}{}' r'$ et $h(s') = s$ et $h(r') = r'$. Soit $s'_1, s'_2, \ldots s'_k$ une séquence d'états dans S' obtenue en démarrant à $s'_1 = s'$ et en terminant à $s'_k = r'$, en prenant la séquence d'actions $g(a)$. Par hypothèse, pour chaque paire (s'_i, s'_{i+1}) dans cette séquence, il existe une paire (s''_i, s''_{i+1}) dans les états du modèle C telle que $h'(s''_i) = s'_i$, $h'(s''_{i+1}) = s'_{i+1}$ et $s''_i \xrightarrow{g'(g(a)_i)}{}'' s''_{i+1}$. En étendant cette analyse à l'ensemble des paires successives dans la séquence $s'_1, s'_2, \ldots s'_k$, on en déduit qu'il existe $s''_i, r''_i \in \mathrm{dom}(h'')$, tels que $s''_i \xrightarrow{g''(a)}{}'' s''_k$, avec $h''(s''_i) = s$ et $h''(s''_k) = r$, ce qui implique que g'' et h'' vérifient la première clause de notre définition.

Maintenant, soit $s \in S$, $a \in \Lambda$, $s'' \in \mathrm{dom}(h'')$ et $r'' \in S''$. Supposons $h''(s'') = s$, et $s'' \xrightarrow{g''(a)}{}'' r''$. Pour démontrer que la seconde clause de notre définition s'applique, nous devons montrer que $r'' \in \mathrm{dom}(h)$ et $s \xrightarrow{a} h''(r'')$.

Pour ce faire, nous procédons à la même décomposition du chemin parcouru dans S'. Par hypothèse, et puisque $g''(a)$ se définit comme la concaténation des séquences obtenues par application de g' sur chaque élément de la séquence $g(a)$, décomposons cette concaténation. Soit $s'_1, \ldots, s'_k$ toute séquence de longueur $k = |g(a)|$ qui vérifie $h(s'_1) = s$, et pour tout $i < k$, $s'_i \xrightarrow{g(a)_i}{}' s'_{i+1}$. Par hypothèse, on sait que $s'_k \in \mathrm{dom}(h)$, et que $s \xrightarrow{a} h(s'_k)$. Soit $s''_1, s''_2, \ldots, s''_k$ la séquence d'états dans S'' qui pour tout $i < k$ vérifient $s''_i \xrightarrow{g'(g(a)_i)}{}'' s''_{i+1}$. On peut prouver par récurrence en démarrant par l'hypothèse $h'(s''_1) = s'_1$, et par hypothèse de la seconde clause de notre définition, que pour tout i, $s''_i \in \mathrm{dom}(h')$, $h'(s''_i) = s'_i$ et $s'_i \xrightarrow{g(a)_i}{}' h'(s''_{i+1})$. En mettant tout cela bout à bout, on obtient que $s''_k \in \mathrm{dom}(h')$, ce qui avec le fait que $h'(s''_k) = s'_k$ et que $s'_k \in \mathrm{dom}(h)$ implique que $s''_k \in \mathrm{dom}(h'')$, et que $s'_1 \xrightarrow{g(a)}{}' h'(s''_k)$, ce qui implique à son tour que $s \xrightarrow{a} h''(s''_k)$. Notre seconde clause est donc également vérifiée. □

Propriété 4. *La simulation en temps polynomial vérifie*
- *la réflexivité : tout modèle A vérifie $A \preccurlyeq_P A$;*
- *la transitivité : pour tous modèles A, B, C, si $A \preccurlyeq_P B$ et $B \preccurlyeq_P C$, alors $A \preccurlyeq_P C$.*

Démonstration. La preuve est analogue au cas logarithmique en observant que la composition de deux fonctions calculables en temps polynomial est également calculable en temps polynomial. □

La simulation linéaire englobe toute simulation dont la construction se limite à la mise à l'échelle du modèle simulé par une certaine constante. Cela inclut la simulation de la règle 184 par la règle 134. Il est cependant nécessaire d'observer que les systèmes de transition d'états définis dans l'exemple 24 qui correspondent aux automates cellulaires élémentaires ne peuvent vérifier aucune simulation mesurée en temps ; cela vient du fait qu'ils définissent des configurations infinies. Pour résoudre ce problème, nous définissons la variation suivante.

Exemple 25. *Pour n un entier, l'ensemble de configurations $\mathbb{B}^n$, l'ensemble d'actions $\{a\}$, et k un entier entre 0 et 255, on définit $\rightarrow_{ECA(n,k)}$ la relation qui vérifie $s \rightarrow s'$ si et seulement si s' s'obtient par mise à jour de l'automate cellulaire élémentaire numéro k sur la configuration finie s. Le triplet $R_{n,k} = (S = \mathbb{B}^n, \Lambda = \{a\}, \rightarrow_{ECA(k)})$ est le système de transition d'états qui correspond à l'automate cellulaire élémentaire numéro k sur des configurations finies périodiques de taille n.*

Pour rappel, l'exécution d'un automate cellulaire élémentaire sur une configuration finie est définie en section 1.7. Cette définition permet la vérification de la simulation illustrée en figure 2.1.

Exemple 26. *Prouvons $R_{n,184} \preccurlyeq_L R_{2n,134}$ pour tout entier n.*

Soit $h : \mathbb{B}^{2n} \twoheadrightarrow \mathbb{B}^n$ la fonction qui est définie sur toute configuration obtenue par la concaténation des mots 00 *et* 01*. La fonction h retourne alors une configuration de taille n obtenue par le remplacement des mots* 00 *par* 1 *et* 01 *par* 0*. On définit $g : \{a\} \to \{a\}^*$ la fonction qui vérifie $g(a) = aa$.*

Ces fonctions réalisent la simulation illustrée en figure 2.1, et sont calculables en espace logarithmique, puisque la fonction g est constante et que la fonction h est définie par une simple opération de réécriture. De plus, pour tout $s \in \mathbb{B}^n$, on peut construire $s' \in \mathbb{B}^{2n}$ tel que $h(s') = s$ en espace logarithmique simplement par l'application de la règle de réécriture $0 \mapsto 01, 1 \mapsto 00$.

Cette propriété est satisfaisante pour décrire cette simulation dans le contexte des systèmes de transition d'états.

2.4 Travail futur

Si la définition présentée dans ce chapitre nous semble prometteuse, nous mentionnons que les propriétés exposées dans ce chapitre ne sont pas suffisantes pour justifier la nature généralisatrice de cette définition. En effet, seul un travail d'application systématique de cette définition aux nombreux travaux du domaine du calcul naturel permettra de se convaincre de sa nature généralisatrice. Au cours de ce travail, et si l'on suppose que notre définition soit assez généralisante, il sera sans doute mis en lumière que les définitions de simulations exprimées pour les différents modèles de calcul naturel possèdent des variations et des subtilités qui préviennent une simple généralisation. Parmi ces subtilités, il est probable que la définition de simulation intrinsèque ne soit pas généralisable dans l'absolu. Bien que la caractérisation de la simulation intrinsèque comme une simulation particulièrement efficace soit intuitive, il reste à savoir si une limite précise de complexité permet cette caractérisation. Peut-être cette étude permettra-t-elle d'exhiber des exemples contre intuitifs d'une simulation acceptée comme intrinsèque mais ayant une plus grande mesure de complexité qu'une autre simulation non intrinsèque ?

Quelles que soient les réponses à ces questions, les définitions proposées dans ce chapitre sont des fragments incomplets qui demandent un travail approfondi pour en cerner les limites. Bien que nous n'ayons aucune garantie à ce stade que notre définition fonctionne dans ce cadre étendu, nous avons confiance que la direction générale de cette définition offre une direction pertinente. À la clé de cette recherche réside la création d'une hiérarchie formelle des modèles de calcul naturel qui permettra peut-être de donner une unité entre les domaines d'horizons très variés qui les régissent.

3 Simulation : développements et discussion

3.1 Simulation et complexité

La complexité est une théorie qui repose sur la classification de problèmes en différentes classes. Le type de problème le plus commun et le plus étudié est le problème de décision. Un tel problème pose une question à laquelle on doit répondre soit par "oui", soit par "non". L'ensemble des questions que l'on peut poser sont les instances du problème, qui sont dites positives lorsque la réponse est "oui", et négatives lorsque la réponse est "non". En général, les instances sont considérées comme des nombres plutôt que des phrases ; il est simple de voir que l'on peut toujours transformer l'un en l'autre.

Nous faisons dans ce chapitre la comparaison entre la réduction et la simulation. Ces deux opérations visent à démontrer que l'on peut résoudre un objet en en utilisant un autre, en permettant une mesure de complexité. Nous sommes intéressés à pousser l'analogie le plus loin possible, pour comprendre quelles structures sont conservées dans la traduction, et lesquelles sont transformées. Nous supposons la définition de simulation développée en section 2.3.2 car, bien qu'elle soit incomplète, elle reste la définition la plus satisfaisante. Pour rappel, la définition de réduction est donnée en section 1.8, accompagnée des autres définitions de base liées à la théorie de la complexité.

3.1.1 Problèmes et systèmes de transition d'états

La première différence entre réduction et simulation est que les objets sur lesquels ces notions opèrent sont fondamentalement différents : les problèmes disposent toujours d'un nombre infini d'instances, et la difficulté de ces problèmes repose uniquement sur la répartition des réponses positives à ces instances. Un système de transition d'états est bien plus libre dans sa structure ; un tel système peut à priori être de toute taille, même finie, et peut inclure des chemins infinis lorsqu'il est infini.

Nous observons ici qu'il existe une façon simple de transformer tout problème de décision en système de transition d'états.

Définition 12. *Soit A un problème de décision disposant d'un ensemble d'instances I ; la transformation de ce problème est un système de transition d'états que nous notons* $\mathrm{STE}(A)$ *qui dispose des états $I \cup \{o, p\}$ où o correspond à la réponse "oui" et p correspond*

à la réponse "non". L'ensemble des actions Λ de ce système est l'ensemble des nombres entiers, que l'on encode en unaire. Pour tout instance i du problème A, on a $i \xrightarrow{n} o$ si et seulement si i est une instance positive de taille n, et $i \xrightarrow{n} p$ si et seulement si i est une instance négative de taille n.

Exemple 27. *On considère SAT le problème de décision qui repose sur la satisfiabilité d'une formule booléenne. Son ensemble d'instances I est l'ensemble des formules booléennes, qui sont définies en section 1.5. Le système de transition d'états $\mathtt{STE}(A_{SAT})$ est composé des états $I \cup \{o, p\}$, soit l'ensemble de toutes les formules booléennes, et les états spéciaux o et p. Pour toute instance i encodable en n bits, ce système admet $i \xrightarrow{n} o$ si i est une formule satisfiable. Si i est insatisfiable, alors le système admet $i \xrightarrow{n} p$.*

Nous utilisons les actions de ces systèmes pour transmettre à la fonction g la taille de l'instance. Cela permet à la fonction g de décider du temps nécessaire à attendre pour résoudre une instance donnée. Cette façon de traduire un problème de décision peut paraître artificielle, et témoigne des différences entre problèmes de décision et systèmes de transition d'états. Pour traduire les problèmes en systèmes, certaines technicités sont nécessaires.

3.1.2 Classes de simulation

La définition des classes de complexité repose sur la résolution de problèmes par des machines limitées en temps et en espace. Nous construisons une analogie basée sur la simulation en remplaçant la condition de résoudre le problème par une machine par la condition que le système étudié soit simulable par la même machine disposant des mêmes restrictions.

Pour permettre cette traduction, nous décrivons les systèmes de transition d'états qui correspondent aux machines de Turing, dont la définition est donnée en section 1.6.

Définition 13. *Pour $T = (S, \Lambda, s_0, \delta, F)$ une machine de Turing déterministe, $\mathtt{STE}(T)$ est le système de transition d'états qui admet les états $S \times \Lambda^{\mathbb{Z}} \times \mathbb{Z}$, l'ensemble d'actions $\{a\}$ et la relation de transition $\rightarrow_T$ telle que, pour toute paire d'états (s, R, z), (s', R', z'), $(s, R, z) \rightarrow_T (s', R', z')$ si et seulement si $\delta(s, R_z) = (s', R'_{z'}, z' - z)$ et $R_k = R'_k$ pour tout $k \neq z$.*

Pour (s, R, z) un état de $\mathtt{STE}(T)$, R représente l'état du ruban, s l'état de la machine et z la position de la tête de lecture. Cette définition est également généralisable au cas non déterministe.

Définition 14. *Pour $T = (S, \Lambda, s_0, \Delta, F)$ une machine de Turing non déterministe, $\mathtt{STE}(T)$ est le système de transition d'états qui admet les états $S \times \Lambda^{\mathbb{Z}} \times \mathbb{Z}$, l'ensemble d'actions $\{a\}$ et la relation de transition $\rightarrow_T$ telle que, pour toute paire d'états (s, R, z), (s', R', z'), $(s, R, z) \rightarrow_T (s', R', z')$ si et seulement si $(s, R_z, s', R'_{z'}, z' - z) \in \Delta$ et $R_k = R'_k$ pour tout $k \neq z$.*

Nous faisons l'observation importante que, telle que décrite dans ces définitions, chaque configuration de ces systèmes de transition d'états contient l'information d'un vecteur bi-infini dans $\Lambda^{\mathbb{Z}}$. Cette observation est problématique puisque ce vecteur passe par définition en entrée et sortie des fonctions qui composent une simulation polynomiale. Si ces entrées et sorties sont infinies, la question d'une mesure de complexité devient inapplicable. Pour contourner ce problème, nous considérons que ce ruban peut être alternativement défini comme la liste des positions dont l'état n'est pas 0, pour 0 un état arbitraire de Λ. Cet encodage permet de définir la simulation d'un modèle à description finie par une machine de Turing, et de mesurer cette simulation en complexité sous l'hypothèse que les configurations employées dans cette simulation comportent toutes un nombre fini de positions dont la valeur n'est pas 0. Il n'est cependant pas difficile de voir que cette hypothèse est conservée dans l'ensemble des simulations considérées dans ce chapitre.

Ces définitions de systèmes de transitions d'états permettent la définition intuitive des classes $\mathsf{P}_{\preccurlyeq}$ et $\mathsf{NP}_{\preccurlyeq}$ suivante.

Définition 15 (La classe $\mathsf{P}_{\preccurlyeq}$)**.** *La* classe $\mathsf{P}_{\preccurlyeq}$ *est l'ensemble des systèmes de transition d'états* $(S, \Lambda, \rightarrow)$ *pour lesquels il existe une machine de Turing déterministe* T *telle que* $(S, \Lambda, \rightarrow) \preccurlyeq_L \mathtt{STE}(T)$.

Définition 16 (La classe $\mathsf{NP}_{\preccurlyeq}$)**.** *La* classe $\mathsf{NP}_{\preccurlyeq}$ *est l'ensemble des systèmes de transition d'états* $(S, \Lambda, \rightarrow)$ *pour lesquels il existe une machine de Turing non déterministe* T *telle que* $(S, \Lambda, \rightarrow) \preccurlyeq_P \mathtt{STE}(T)$.

Naturellement, la définition de ces classes permet de vérifier que la classe $\mathsf{P}_{\preccurlyeq}$ est incluse dans la classe $\mathsf{NP}_{\preccurlyeq}$.

Propriété 5.

$$P_{\preccurlyeq} \subseteq NP_{\preccurlyeq}$$

Démonstration. On observe que toute machine déterministe $T = (S, \Lambda, s_0, \delta, F)$ est également la machine non déterminisite $(S, \Lambda, s_0, \Delta, F)$ telle que $(s, x, s', y, p) \in \Delta$ si et seulement si $\delta(s, x) = (s', y, p)$. De plus, toute simulation en espace logarithmique est également une simulation en temps polynomial. On obtient donc que tout système de transition d'états inclus dans $\mathsf{P}_{\preccurlyeq}$ est également inclus dans $\mathsf{NP}_{\preccurlyeq}$. □

La comparaison entre ces classes de simulation et les classes de complexité ne s'arrête pas là. Nous montrons qu'un problème A est dans P si et seulement si sa traduction $\mathtt{STE}(A)$ est dans $\mathsf{P}_{\preccurlyeq}$.

Propriété 6. *Soit A un problème de décision. Si $A \in P$, $\mathtt{STE}(A) \in P_{\preccurlyeq}$.*

Démonstration. Par hypothèse, il existe une machine de Turing T qui résout le problème A. Pour prouver la simulation, nous définissons la fonction h comme définie sur le domaine des configurations d'une machine de Turing qui est dans un état initial (et encode une instance de notre problème), ou final. Ainsi $h(s')$ vaudra i si la

configuration s' est dans l'état initial et contient l'instance i du problème A. Sinon, $h(s')$ vaudra o si la machine est dans un état final acceptant, et p si la machine est dans un état final de refus. Cette fonction est calculable en espace logarithmique ; d'abord, dans le cas d'un état final , la fonction est calculable en temps constant. Dans le cas d'une configuration initiale, on considère que h répond simplement l'instance rédigée dans le ruban de la machine dans son état initial, ce qui est clairement calculable en espace logarithmique. Cela fonctionne car on considère que toute instance mal formée du problème A est une instance négative du problème A. Grâce à cette hypothèse, h admet toute configuration initiale de la machine dans son domaine, et chaque instance de A est transformée en exactement une configuration initiale. Autrement dit, la charge de la preuve que i est une instance bien formée est laissée à la machine de Turing T plutôt qu'à h.

Par hypothèse que $A \in \mathsf{P}$, il existe un polynôme f qui pour toute taille d'instance n borne le nombre de mises à jour nécessaire pour décider toute instance de taille n grâce la machine qui la résout. On suppose ici que la machine T résout toute instance de taille n en exactement $f(n)$ étapes, puis s'arrête. La fonction g prend en entrée un nombre n égal à la taille de l'instance encodé en unaire, et calcule une séquence définie comme la répétition $a^{f(n)}$. Grâce au fait que n est encodé en unaire, ce processus peut-être opéré en espace logarithmique : on compte d'abord la taille de l'entrée en binaire, puis on applique le polynôme f. On peut déduire que le bit numéro k de la sortie est un a si et seulement si $k < f(n)$.

Il s'en suit que g et h sont toutes les deux calculables en espace logarithmique. On note également que h est surjective, car toute instance correspond à au moins une configuration initiale, et les réponses o et p sont obtenues depuis toute configuration finale.

Soient S, Λ les ensembles d'états et d'actions de $\mathtt{STE}(A)$, et S' l'ensemble des états de $\mathtt{STE}(T)$.

Supposons $s, r \in S$, $a \in \Lambda$ et $s' \in \mathrm{dom}(h)$, tels que $h(s') = s$ et $s \xrightarrow{a} r$. L'état s est nécessairement une instance i du problème A. Alors s' est une configuration initale de machine de Turing qui encode l'instance i, et $a = |i|$. Il existe donc $r' \in \mathrm{dom}(h)$ la configuration qui correspond au résultat du calcul de la machine qui prend i en entrée, telle que $h(r') = r$ (le résultat de la machine est cohérent avec la réponse à l'instance), et $s' \xrightarrow{g(a)} {}' r'$ (par hypothèse la machine trouve ce résultat après exactement $|g(|i|)|$ étapes de temps). La première clause de la simulation est donc vérifiée.

Supposons maintenant $s \in S$, $a \in \Lambda$, et $s', r' \in \mathrm{dom}(h)$ tels que $h(s') = s$ et $s' \xrightarrow{g(a)} {}' r'$. On sait de nouveau que s est une instance i de A, car dans le cas contraire s' serait une configuration finale de la machine, qui n'a pas de successeur. Alors s' est une configuration initiale qui encode cette instance, et r' est donc nécessairement une configuration finale. Par l'hypothèse que la machine résout l'instance en exactement $f(|i|)$ étapes et donc $a = |i|$. On obtient donc que $s \xrightarrow{a} h(r')$, ce qui prouve la seconde clause de la simulation.

□

Propriété 7. *Soit A un problème de décision. Si* $\mathtt{STE}(A) \in P_{\preccurlyeq}$*, alors* $A \in P$.

Démonstration. Par hypothèse, pour toute instance i du problème A, il existe une configuration s' d'une machine de Turing T donnée telle que $h(s') = i$. Après $|g(|i|)| = f(n)$ mises à jour, pour f un polynôme, on obtient une configuration de machine de Turing r' telle que $r' \in \mathrm{dom}(h)$ et $h(r')$ est la réponse à l'instance i. Il est à noter que la configuration r' n'a aucune garantie d'être finale, pas plus que s' n'a de garantie d'être initiale. Cependant comme par hypothèse $h(r')$ peut être calculée en espace logarithmique, nous pouvons construire T' une machine qui se comporte de façon identique à la machine T, à l'exception qu'une fois à la configuration r', T' calcule $h(r')$ et converge vers l'état d'acceptation ou de refus correspondant. Cette machine T' résout alors i en temps polynomial en démarrant depuis la configuration s' modifiée pour que la machine soit alors dans un état initial. □

Il est intéressant qu'un problème P-difficile ne se traduise donc pas en un système $\mathrm{P}_{\preccurlyeq}$-difficile, en supposant que cette dernière notation signifie que le système en question est capable de simuler en espace logarithmique tout autre système dans $\mathrm{P}_{\preccurlyeq}$. Une instance donnée d'un problème de décision n'est capable de prédire l'état d'une machine de Turing qu'après un nombre fini de mises à jour, et ne pourra donc pas simuler les détails d'une exécution potentiellement sans fin. Il existe cependant bien des modèles $\mathrm{P}_{\preccurlyeq}$-difficiles : les modèles $\mathtt{STE}(T)$, pour T toute machine de Turing universelle, en sont des exemples qui découlent directement de la définition de $\mathrm{P}_{\preccurlyeq}$. L'intuition nous invite même à formuler la conjecture suivante. Ici, *efficacement Turing universel* signifie capable de simuler toute machine de Turing de façon à ce que le décodage d'une configuration (calcul de l'état, des symboles inscrits sur le ruban, et de la position de la tête) soit réalisable en espace logarithmique.

Conjecture 1. *L'ensemble des systèmes efficacement Turing universels est l'ensemble des systèmes $P_{\preccurlyeq}$-difficile.*

Essayons maintenant de produire les même résultats pour les relations entre les classes NP, coNP et $\mathrm{NP}_{\preccurlyeq}$. Les problèmes de la classe NP peuvent être décrits de la façon suivante : ils peuvent être résolus en temps polynomial par une machine de Turing non déterministe, c'est-à-dire en particulier que si l'instance étudiée est positive, alors il existe une branche de taille polynomiale dans l'exécution de la machine qui retourne la réponse "oui". Les problèmes de la classe coNP sont similaires, et si une instance étudiée d'un tel problème est négative, alors il existe au moins une branche de taille polynomiale dans l'exécution de la machine qui répond "non". Intuitivement, une instance positive d'un problème dans NP se vérifie facilement dès que l'on dispose du bon indice, et une instance négative d'un problème dans coNP se démontre facilement fausse dès que l'on dispose d'un bon contre-exemple.

La distinction entre ces deux classes est cohérente dans la théorie de la complexité, bien que leur égalité reste un problème ouvert, car la notion de réduction ne permet pas l'inversion de la réponse aux instances. Tout problème dans NP dispose d'un dual dans coNP, où les indices deviennent des contre-exemples. Les problèmes

NP-complets n'admettent pas de réduction polynomiale connue vers leurs duaux respectifs, et l'existence d'une telle réduction impliquerait que NP = coNP. Au meilleur de nos connaissances actuelles, l'égalité entre ces classes est un problème ouvert.

Dans le cas de la simulation telle que nous la définissons, il n'existe pas de convention pour l'acceptation ou le refus d'une instance. Cela semble logique ; dans le cas le plus général, les modèles de calcul n'ont pas la vocation de résoudre des problèmes, où même la vocation de faire quoi que ce soit en particulier. Un automate cellulaire qui évolue sur une configuration infinie va évoluer sur une durée de temps infinie, et aucune convention de résultat d'un calcul ne ferait sens en dehors d'un contexte très précis. Nous faisons même la remarque que c'est le rôle d'une définition générale de la simulation de permettre l'application d'un tel contexte sur le calcul, et en particulier de définir les conventions qui définissent la validité ou l'invalidité d'une instance par exemple dans le cas d'une simulation d'un problème de décision par la règle 110, qui est un modèle de calcul universel. De cette impossibilité de distinguer les conventions de validation et de refus ressort le fait suivant : il n'existe pas de traduction de NP et coNP qui permette de les distinguer dans le contexte de la simulation.

Le problème est encore plus profond. La définition des classes NP et coNP permet de définir la façon dont les problèmes concernés doivent être résolus. Ainsi, l'instance d'un problème dans NP est positive s'il existe une branche de calcul qui répond "oui" dans la machine de Turing correspondante, ce qui correspond à notre définition de simulation. Cependant, notre définition de simulation, qui ne traite pas le "oui" de façon différente du "non", va également interpréter une instance comme négative s'il existe une branche de calcul qui répond "non".

Pour contourner ce problème, nous proposons de conserver $\mathsf{NP}_{\preccurlyeq}$ comme une classe englobant les traductions des problèmes de NP et coNP. Nous faisons la distinction par la façon de traduire le problème de décision concerné en un système de transition d'états.

Définition 17 (Système positif d'un problème de décision)**.** *Soit A un problème de décision, avec un ensemble d'instances I. Le système de transition d'états positif de A, noté $\mathtt{STE}^+(A)$, est défini sur les nœuds $I \cup \{o\}$. Son ensemble d'actions est l'ensemble des entiers encodés en unaire. Pour toute instance $i \in I$, ce système admet $i \xrightarrow{n} o$ où n est la taille de l'instance i.*

Définition 18 (Système négatif d'un problème de décision)**.** *Soit A un problème de décision, avec un ensemble d'instances I. Le système de transition d'états négatif de A, noté $\mathtt{STE}^-(A)$, est défini sur les nœuds $I \cup \{p\}$. Son ensemble d'actions est l'ensemble des entiers encodés en unaire. Pour toute instance $i \in I$, ce système admet $i \xrightarrow{n} p$ où n est la taille de l'instance i.*

Cette redéfinition de la traduction d'un problème de décision nous permet de nous concentrer sur les instances qui sont considérées importantes pour les définitions des classes qui nous intéressent.

Propriété 8.

$$A \in NP \iff \mathtt{STE}^{+}(A) \in NP_{\preccurlyeq}$$
$$A \in coNP \iff \mathtt{STE}^{-}(A) \in NP_{\preccurlyeq}$$

Démonstration. Pour prouver que $A \in \mathsf{NP} \implies \mathtt{STE}^{+}(A) \in \mathsf{NP}_{\preccurlyeq}$ et $A \in \mathsf{coNP} \implies \mathtt{STE}^{-}(A) \in \mathsf{NP}_{\preccurlyeq}$, nous observons une simulation identique à la simulation définie dans la preuve de la propriété 6. Le résultat découle du fait que la définition précise des $\mathtt{STE}^{+}(A)$ et $\mathtt{STE}^{-}(A)$ requiert que la simulation ne couvre que les cas qui sont garantis par les hypothèses $A \in \mathsf{NP}$ et $A \in \mathsf{coNP}$ respectivement. Pour prouver que $\mathtt{STE}^{+}(A) \in \mathsf{NP}_{\preccurlyeq} \implies A \in \mathsf{NP}$ et $\mathtt{STE}^{-}(A) \in \mathsf{NP}_{\preccurlyeq} \implies A \in \mathsf{coNP}$, nous observons un ensemble d'arguments similaires à la preuve de la propriété 7. □

La méthode employée pour obtenir ce résultat est questionnable : il ne serait pas raisonnable de définir une façon différente de perçevoir les problèmes de décision à chaque nouvelle classe que nous voudrions reproduire. Cette solution *ad hoc* est en vérité le témoin d'une différence importante entre la théorie de la complexité et l'extension que nous tentons. La théorie de la complexité fait la distinction entre la réduction, qui permet la comparaison entre problèmes, et la résolution d'un problème par une machine, qui est fondamentale pour la définition des classes de complexité. Dans le cadre que nous étudions, ces deux notions sont interprétées par la simulation. L'élégance éventuelle de cette manœuvre retire la liberté de définir les classes par la façon spécifique dont la machine résout le problème. Pour en reproduire la structure, nous sommes contraints de reproduire cette spécificité là où cela est possible, spécifiquement dans la traduction des problèmes de décision que nous admettons.

Il est important de rappeler que les notions de réduction et de simulation restent fondamentalement différentes car définies sur des objets différents. Si nous trouvons encourageant de trouver des similarités entre les structures qui sont obtenues par la théorie de la complexité et la simulation, il est non moins encourageant de comprendre les structures des classes de simulation pour ce qu'elles sont, et en particulier en la façon dont elles divergent de notre compréhension. Car, bien que ce manuscrit ne se concentre que sur l'adaptation d'un fragment réduit de la théorie de la complexité dans le contexte de la simulation comme d'un gage de cohérence, la simulation permet l'inclusion de bien plus de formes de calcul, qui dépassent la cadre habituel de la complexité.

3.2 Intuition générale de la simulation

Lorsque nous étudions des modèles mathématiques, nous discutons souvent des intuitions qui se cachent derrière les définitions. La simulation n'est pas en reste ; cette notion, qui a été formellement développée indépendamment dans de nombreux domaines, nous encourage à la généraliser, par l'intuition que si l'ensemble des domaines concernés ont employé ce terme pour parler de choses indépendantes, c'est

qu'il existe une perception commune de ce qu'est la simulation. Si cela est avéré, cela nous encouragerait à poursuivre cette perception dans notre recherche d'une définition générale. Quelle est cette perception ? Pour conclure notre travail sur la simulation, nous proposons dans cette section une forme de discussion sur les idées qui entourent la simulation. Nous essayons, informellement, de comprendre ce qui lie la notion de simulation à la notion informelle de calcul ; et nous faisons des liens avec l'idée d'émergence.

Ce discours informel ne cherche pas à permettre l'élaboration d'un quelconque théorème, ou de tout autre résultat formel mesurable. Il cherche plutôt à organiser un ensemble d'images afin de clarifier notre propre perception du terme de simulation, et peut-être aussi d'élaborer des outils propres à l'éducation et la vulgarisation des notions employées.

3.2.1 La simulation et la notion de calcul

Quelle est l'intuition commune derrière la simulation ? Dans la pratique, la simulation permet de montrer qu'un modèle peut calculer tout ce qu'un autre modèle calcule. Il semble donc inévitable que si l'on veut parler de simulation, on doit également parler de calcul.

Le calcul est une notion qui dispose d'énormément d'exemples formels ; ils sont aussi nombreux que les modèles de calcul qui en portent le nom, mais aussi les très nombreux algorithmes et machines concrètes qui parcourent les domaines appliqués de l'informatique. Il est pourtant difficile d'extraire de tous ces exemples l'ensemble des critères formels qui établissent que quelque chose "calcule". Le travail qui semble se rapprocher le plus de cette formalisation est la thèse de Church-Turing, qui stipule l'existence de modèles aux calculs universels. Cependant, si elle nous donne un exemple de modèle universel, la thèse de Church-Turing ne permet pas de trouver les critères qui font qu'un modèle est universel.

Nous posons la notion du calcul de la façon suivante : un calcul est un processus qui permet la résolution d'un problème. Ainsi, si un automate cellulaire élémentaire (voir section 1.7) ne possède pas la fonction première de résoudre un problème pratique, la littérature qui s'y intéresse mesure la capacité de calcul des différentes règles élémentaires par leur capacité à résoudre des problèmes. Par le biais de la simulation illustrée en figure 2.1, on exprime que la règle 134 possède une capacité de calcul au moins aussi puissante que la règle 184 ; par la simulation développée par (Cook 2004), on sait que la règle 110 est capable de résoudre tout problème qu'une machine de Turing peut résoudre et, par conséquent, que son calcul est au moins aussi puissant que ces dernières.

Quelle est la fonction de la simulation dans le calcul ? La mention de simulation dans le précédent paragraphe met en avant la fonction première de la simulation en pratique : prouver la capacité de calcul d'un modèle en se servant du point de référence d'un autre modèle. Peut-on trouver une description plus absolue ?

Fondamentalement, il n'existe pas de problème sans intelligence pour se le poser.

On en vient donc au rôle de l'observateur dans la notion de calcul, et plus précisément, la façon dont le rôle de l'observateur sublime celui de la simulation dans notre relation humaine au calcul.

3.2.2 La simulation, outil pratique de traduction

Pour guider le développement de cette idée, nous proposons la série d'images suivantes.

Alice dispose d'un problème en tête. Il s'agit d'un problème arithmétique. Pour le résoudre, Alice rédige les équations pertinentes sur le papier et trouve la solution.

Bob dipose d'un problème arithmétique, et pour le résoudre dispose d'une calculatrice. Bob entre le problème dans la calculatrice et la calculatrice lui donne une réponse, que Bob recopie.

Charlie est abandonné sur une planète inconnue, et souhaite résoudre un problème arithmétique. Charlie est devant une machine extra-terrestre inconnue. Fort heureusement, Charlie dispose également d'un manuel d'utilisation, qui permet de se servir de cette machine inconnue comme d'une calculatrice. Charlie traduit son problème en quelque chose que la machine peut comprendre, puis interprète le résultat en quelque chose que lui même comprend, le tout grâce au manuel.

Dans ces trois histoires, chacun de nos acteurs effectue un calcul ; dans quelle histoire a-t-on un exemple de simulation ? L'histoire de Charlie est construite pour contenir la simulation la plus explicite. C'est le manuel d'instruction qui sert de preuve explicite que la machine extra-terrestre est capable du même calcul qu'une calculatrice. Est-ce la seule simulation que l'on peut extraire de ces histoires ?

En plus des machines concrètes décrites dans ces histoires – la calculatrice, la machine extra-terrestre – Alice, Bob et Charlie disposent d'un problème d'arithmétique qui réside dans leur tête. Ce problème peut être résolu ; et par notre définition de calcul, tout ce qui permet de le résoudre est alors un objet qui calcule. Cependant, nous estimons que ce problème est aussi lui-même un objet qui calcule car, à l'instar des problèmes de décision mentionnés en section 3.1, il est capable de prendre part dans une réduction avec d'autres problèmes, ou d'être simulé par des modèles. Les histoires d'Alice et Bob sont donc également une histoire de simulation, où la résolution de leur problème présuppose une simulation de l'ensemble des problèmes arithmétiques par les équations et la calculatrice respectivement. Cela est également observable dans l'histoire de Charlie, qui décrit finalement une simulation en deux étapes. Bien que Charlie ne dispose pas de calculatrice, mais seulement de la machine extra-terrestre, le manuel d'utilisation lui permet d'utiliser la complexité de la machine extra-terrestre comme d'une calculatrice : le manuel d'utilisation constitue en vérité une preuve de simulation d'une calculatrice par la machine extra-terrestre. En plus de cette simulation, Charlie utilise cette calculatrice virtuelle pour résoudre son problème. Et comme pour Alice et Bob, cela constitue en soi une simulation, qui est la seconde étape de la résolution du problème de Charlie.

Nous disons donc la chose suivante : dans tout emploi du calcul par un observateur

pour résoudre un problème, il réside une simulation parfois implicite qui repose sur la distance intrinsèque entre le problème posé et la machine qui, par la simulation, le résout. Par cette proposition, la simulation devient plus qu'un outil permettant la démonstration qu'un modèle est aussi expressif qu'un autre, mais également une condition nécessaire à l'emploi d'une machine pour toute tâche qui nécessite une interprétation, même triviale, pour sa résolution. Dans l'histoire de Bob, par exemple, l'utilisation du clavier de la calculatrice et la lecture de son écran sont des étapes nécessaires d'une interprétation simple de son calcul.

Que signifie cette observation ? Principalement que la simulation dépasse intuitivement le cadre spécifique des modèles de calcul naturel, et nous parle plus largement de notre rapport au calcul en général, qui est fondamental subjectif.

Pour mettre en avant ce dernier point, considérons de nouveau l'histoire de Charlie. Supposons que, dans une variante de cette histoire, Charlie ne dispose pas de ce manuel d'utilisation qui lui permet l'utilisation de la machine extra-terrestre. Maintenant, du point de vue de Charlie, la machine n'est plus capable de résoudre son problème d'arithmétique. Charlie pourra observer que le comportement de cette machine est complexe, mais en l'absence d'une façon concrète de faire sens de cette complexité, peut-on se convaincre que cette machine calcule ? Cette situation est analogue à beaucoup de positions de la littérature sur des modèles de calcul dont la capacité de calcul est supposée mais pas démontrée, comme l'automate cellulaire élémentaire 54 par exemple.

Cette histoire nous invite à réfléchir à l'espace entre l'observateur et la machine. Cet espace implique un travail de l'observateur pour interpréter sa machine, travail que nous nommons simulation. Si la machine extra-terrestre dispose d'une capacité intrinsèque à résoudre des problèmes, la simulation est le procédé qui permet de reconnaître et d'exploiter cette capacité. La simulation semble bien se définir de façon conjointe au calcul ; la simulation est la partie du calcul qui implique la traduction d'énoncés ou de solutions d'un modèle vers un autre.

3.2.3 Simulation et émergence

Dans l'histoire de Charlie, nous touchons à une idée très intéressante : l'idée que sans avoir la clé pour comprendre le calcul d'une machine complexe, nous sommes tout de même capables d'en percevoir superficiellement la complexité. En faisant l'observation d'un modèle tel que l'automate cellulaire élémentaire 54, nous faisons le constat que des comportements complexes émergent.

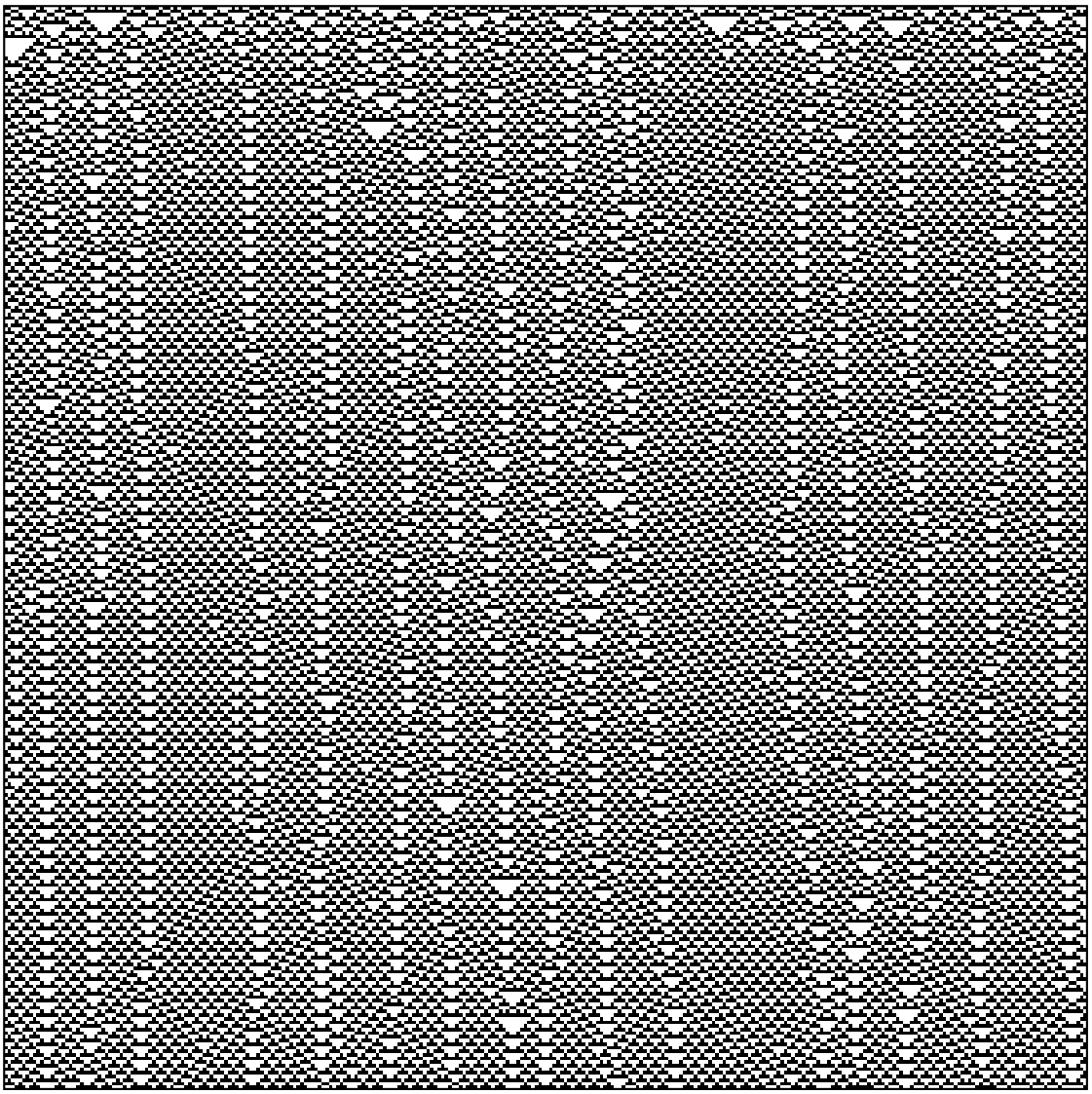

FIGURE 3.1 – Représentation d'une exécution de la règle 54. Cette règle est connue pour ses comportements complexes. D'une configuration initiale aléatoire émergent des structures qui s'apparentent à des particules qui évoluent, se déplacent et interagissent.

Qu'est-ce que l'émergence ? C'est une notion qui, historiquement, repose sur l'observation que, bien que l'univers semble régi par des règles fixées dont nous connaissons certains détails, il existe des choses dans cet univers d'une complexité telle qu'il est (à cet instant du moins) impossible de les déduire de ces règles élémentaires ; l'existence de la conscience humaine, quelle que soit sa définition, est un exemple commun de phénomène pour lequel une description complète apparaît hors de portée. Ceci engendre notamment la perte du lien causal intuitif entre les règles fondamentales et

les phénomènes observés. On dit de ces phénomènes complexes qu'ils *émergent* dans le système.

La question de définir la nature précise de cette émergence est une question difficile (LEWES 1875 ; O'CONNOR 1994 ; CRUTCHFIELD 1994). Parmi les définitions les plus populaires de l'émergence, on citera deux exemples : l'*émergence forte* affirme que les formes qui émergent à un niveau supérieur, c'est-à-dire les phénomènes complexes, possèdent une influence sur les composantes du niveau inférieur, c'est-à-dire les éléments les plus simples du système. Cette perspective est rarement acceptée par le point de vue scientifique, car elle évoque des positions controversées telles que le vitalisme (l'idée que le vivant ne se réduit pas à des règles physico-chimiques). L'*émergence faible* affirme que, bien que les phénomènes émergents fassent preuve d'une complexité qui justifie leur étude par la science, ces phénomènes sont bel et bien le fruit des règles élémentaires du système. Cette position fait la supposition que si, à un moment donné, nous échouons à décrire un phénomène à partir des règles fondamentales, il s'agit avant tout de la faute de notre compréhension incomplète de la réalité.

Il existe des travaux qui discutent de la relation entre émergence et simulation (BOSCHETTI 2012 ; WILDMAN et SHULTS 2018), mais la simulation dont il est question ici n'est pas la simulation employée dans ce chapitre, il s'agit plutôt de la sorte de simulation qui permet, par exemple, de vérifier une théorie fondamentale de la physique sur de puissantes machines. Dans cette section, nous sommes intéressés à tisser les liens entre émergence et simulation, principalement en suivant l'idée que l'émergence se définit souvent comme dépendante de la perspective (CRUTCHFIELD 1994). Cette notion signifie que notre proportion à décrire un phénomène comme émergent dépend principalement de notre compréhension de ce dernier. Moins nous le comprenons, et plus il nous semble justifié de le percevoir comme émergent. Plus nous le comprenons, et plus il nous semblera juste de le décrire comme une conséquence entendue de l'application des règles fondamentales.

Nous invitons à repenser à l'histoire de Charlie et de sa machine extra-terrestre, et de la façon dont, pour Charlie, la capacité de calcul de cette machine dépend de sa compréhension de cette dernière. Lorsque cette compréhension est absente, Charlie voudra exprimer la complexité apparente de cette machine comme un comportement émergent. Le manuel d'utilisation lui confère le savoir qui lui est nécessaire pour faire partiellement sens de cette complexité ; à l'aide de ce manuel, la machine cryptique devient, entre autres, une calculatrice, dont le fonctionnement est intuitivement compris.

La simulation, incarnée dans cette histoire par le manuel d'utilisation, devient alors l'incarnation formelle de l'explication qui permet de réduire certains comportements à d'autres.

Lorsque nous observons un phénomène complexe incompris d'un système de calcul, les éléments développés dans cette section invitent à établir qu'il est raisonnable de décrire ce phénomène comme émergent. Il semble également clair que, dès que la perception d'une émergence est expliquée par une simulation, le phénomène se

range alors dans la catégorie de l'émergence faible. La simulation, par sa notion très générale, semble être le représentant parfait des procédures qui nous permettent de comprendre et d'expliquer ce phénomène en le réduisant à d'autres phénomènes mieux compris. Son rôle vis-à-vis du calcul est en vérité fondamentalement epistémologique, c'est-à-dire que toute connaissance d'un calcul est apportée par une simulation. Plutôt que de créer du calcul, la simulation découvre, explique et détaille le calcul d'un modèle en le comparant au calcul d'un autre modèle.

3.2.4 Conclusion

La simulation touche, grâce au domaine du calcul naturel, aux intuitions qui appartiennent à de très nombreuses sciences comme la biologie ou la physique. La position unique de cette notion très générale fait partie des raisons qui nous poussent à mieux cerner l'intuition qui régit ses très nombreuses apparitions dans les domaines du calcul naturel.

Sur le plan formel, la simulation est un outil largement utilisé, sous diverses formes, pour décrire la complexité – dans le sens large du terme – des très nombreux modèles du calcul naturel. Sur le plan intuitif, il s'agit d'une notion dont les racines sont profondes et touchent aux relations fondamentales entre l'observateur et le calcul. Ces deux aspects font de la question de la généralisation de la simulation un projet ambitieux et dont les enjeux sont majeurs. Bien que le travail présenté dans ce manuscrit pose bien plus de questions qu'il n'apporte de réponses, nous espérons transmettre l'importance de cette direction particulière et chercherons à la développer dans le futur.

4 Réseaux d'automates

4.1 Introduction

Les réseaux d'automates tiennent leur origine dans la théorie des automates à seuils, dont la motivation initiale était d'imiter les réseaux de neurones observés dans le vivant (McCulloch et Pitts 1943). Bien que cette perspective ait donné naissance à la théorie des automates (Kleene et Post 1954 ; Elspas 1959 ; Golomb et al. 1967) et que la perpective en direction de la cognition ait amené aujourd'hui au domaine productif des réseaux de neurones dans l'apprentissage automatique , les réseaux d'automates – définis comme une vision plus large des automates à seuil et des réseaux de neurones – restent un sujet d'étude actif.

L'étude des réseaux d'automates est movitée par leur application en biologie dans le contexte de l'étude de réseaux de régulation de gènes (Kauffman 1969 ; Thomas 1973 ; Demongeot, Goles, Morvan et al. 2010 ; Mendoza et Alvarez-Buylla 1998 ; Davidich et Bornholdt 2008). Dans ces réseaux, chaque automate représente un gène et la façon dont l'automate est mis à jour représente la façon dont les autres gènes l'influencent, que ce soit par activation ou par inhibition. Lorsque l'on considère l'interaction d'un ensemble de ces gènes, le comportement du système devient difficile à prédire : cela vient du nombre exponentiel de configurations possibles du système par rapport à sa taille. Le domaine des réseaux d'automates cherche donc principalement des méthodes pour permettre la prédiction de la dynamique d'un réseau sans la nécessité d'en calculer l'entièreté.

Ainsi, la littérature des réseaux d'automates s'est concentrée sur des structures de réseaux précises (Noual 2012 ; Alcolei, Perrot et Sené 2016), et a permis la prédiction de certains aspects de leur dynamique (Aracena 2008 ; Demongeot, Noual et Sené 2012 ; Aracena, Richard et Salinas 2017), ainsi que la description en complexité de questions associées (N. Alon 1985 ; Floreen et Orponen 1989 ; Orponen 1992 ; Bridoux, Durbec, Perrot et al. 2019 ; Bridoux, Gaze-Maillot, Perrot et al. p. d. ; Ilango, Loff et Oliveira 2020 ; Noûs, Perrot, Sené et al. 2020). Ces travaux de complexité permettent la mesure de la difficulté de la prédiction du comportement limite des réseaux étudiés, par exemple la prédiction de l'existence de points fixes dans leur dynamique.

Dans ce chapitre, nous développons les définitions associées aux réseaux d'automates, à leur mise à jour et à l'étude de leur dynamique.

4.2 Ensembles d'automates, états et configurations

Les *réseaux d'automates* sont définis sur un ensemble d'*automates*. Par convention, nous appellons cet ensemble S. Chaque élément dans S correspond à un automate dans le réseau. Par convention également, les éléments de S sont notés par des lettres minuscules, commençant par a. Bien que, dans le cadre de ce mémoire, les réseaux d'automates soient définis sur des ensembles finis d'automates, il est tout à fait concevable de définir S comme un ensemble infini dénombrable.

Pour un réseau d'automates donné, chaque automate dans S possède un *état*. Le champs des états possibles est défini par l'ensemble d'états de ce réseau d'automates. Cet ensemble est par convention noté Λ. Dans le cas où $\Lambda = \{0,1\}$, le réseau d'automates en question est un réseau d'automates booléens.

Le rassemblement des états de chaque automate du réseau constitue la *configuration* du réseau dans sa globalité. Cette configuration est formellement définie comme un vecteur sur S avec valeurs dans Λ. L'espace de ces vecteurs est noté Λ^S. Nous noterons en général une telle configuration x.

Définition 19 (Configuration)**.** *Soit S un ensemble d'automates et Λ un ensemble d'états. Une* configuration *est un vecteur défini sur Λ^S.*

4.3 Fonctions locales et graphe d'interaction

Étant donné un ensemble d'automates et un ensemble d'états, les *fonctions locales* définies par ces ensembles sont toutes les fonctions qui prennent en entrée la configuration du réseau, et qui retournent un état.

Définition 20 (Fonction locale)**.** *Soit S un ensemble d'automates et Λ un ensemble d'états. Une* fonction locale *est une fonction f définie sur $\Lambda^S \to \Lambda$.*

Enfin, un réseau d'automates définit une fonction locale à chaque automate s dans S.

Définition 21 (Réseau d'automates)**.** *Soit S un ensemble d'automates et Λ un ensemble d'états. Un* réseau d'automates *associe à tout $s \in S$ une fonction locale $f_s : \Lambda^S \to \Lambda$.*

Exemple 28. *Soit $S = \{a,b,c\}$ et $\Lambda = \{0,1\}$. Soit F le réseau d'automates qui à a, b et c associe les fonctions suivantes : $f_a(x) = \neg x_a$, $f_b(x) = x_a \vee x_c$, et $f_c(x) = \neg x_a \wedge \neg x_c$.*

Exemple 29. *Soit $S = \{d,e,g\}$ et $\Lambda = \{0,1\}$. Soit F' le réseau d'automates qui à d, e et g associe les fonctions suivantes : $f'_d(x) = \neg x_g$, $f'_e(x) = \neg x_d$, et $f'_g(x) = x_e$.*

Les réseaux d'automates sont des objets complexes aux interactions nombreuses. Afin d'aider notre compréhension de ces objets, nous utilisons une représentation graphique des réseaux d'automates sous forme de graphe. Dans ce graphe, nommé

graphe d'interaction, chaque automate est un sommet, et il y aura une arête orientée d'un sommet s vers un sommet r si et seulement si la valeur de s dans la configuration influence le calcul de la fonction locale de r. Si c'est le cas, on dit que s *influence* r, ce qui est défini comme suit.

Définition 22 (Influence)**.** *Soit F un réseau d'automates, et $s, r \in S$. On dit que s* influence *r si et seulement s'il existe une paire de configuration x et x' telle que $x_q = x'_q$ pour $q \neq s$ et $f_r(x) \neq f_r(x')$.*

Dans le cas booléen, l'influence peut être définie plus précisément dans le cas d'une influence positive d'un automate par un autre, ou d'une influence négative d'un automate par un autre. Ces notions émergent naturellement de l'application des réseaux d'automates dans l'étude de réseaux de régulation génétique, dans lesquels un gène va généralement permettre l'activation ou l'inhibition d'autres gènes.

Définition 23 (Influence positive)**.** *Soit F un réseau d'automates booléens, et $s, r \in S$ tels que s influence r. On dit que s* influence positivement *r si et seulement si pour toute paire de configuration x et x' telle que $x_q = x'_q$ pour $q \neq s$, on a*

$$x_s < x'_s \iff f_r(x) \leq f_r(x').$$

Définition 24 (Influence négative)**.** *Soit F un réseau d'automates booléens, et $s, r \in S$ tels que s influence r. On dit que s* influence négativement *r si et seulement si pour toute paire de configuration x et x' telle que $x_q = x'_q$ pour $q \neq s$, on a*

$$x_s < x'_s \iff f_r(x) \geq f_r(x').$$

Exemple 30. *Soit $S = \{a, b, c\}$ un ensemble d'automates. Rappellons le réseau d'automates F de l'exemple 28 qui à a, b et c associe les fonctions suivantes : $f_a(x) = \neg x_a$, $f_b(x) = x_a \vee x_c$, et $f_c(x) = \neg x_a \wedge \neg x_c$. Pour étudier l'influence de c sur les automates du réseau, détaillons l'ensemble des paires de configurations x, x' telles que $x_q = x'_q$ pour $q \neq c$:*

$$\{(000, 001), (010, 011), (100, 101), (110, 111)\}.$$

Si l'on remplace chacune de ces configurations x par $f_a(x)$, on obtient

$$\{(1, 1), (1, 1), (0, 0), (0, 0)\}.$$

Si on les remplace par $f_b(x)$, on obtient

$$\{(0, 1), (0, 1), (1, 1), (1, 1)\},$$

et si on les remplace par $f_c(x)$, on obtient

$$\{(1, 0), (1, 0), (0, 0), (0, 0)\}.$$

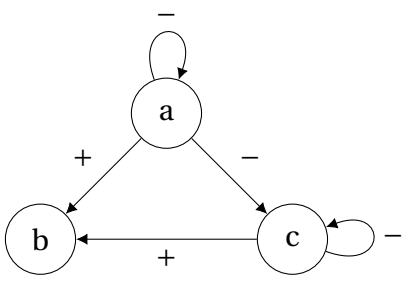

FIGURE 4.1 – Graphe d'interaction du réseau d'automates développé dans l'exemple 28.

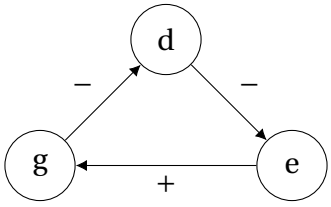

FIGURE 4.2 – Graphe d'interaction du réseau d'automates développé dans l'exemple 29.

On peut voir que c influence b et c, mais pas a. En effet, simplement modifier la valeur de x_c n'a aucun effet sur le calcul de $f_a(x)$. On peut également voir que l'influence de c sur b est positive, et que l'influence de c sur lui même est négative.

Il est possible qu'un automate s influence un automate r, sans pour autant que cette influence soit positive ou négative. Une telle influence est dite *non-monotone*.

Ces relations d'influence nous permettent de décrire le graphe d'interaction de tout réseau. Cette définition nous donne une représentation intuitive de la propagation de l'information à travers le réseau, et ce même graphe sera utilisé comme outil de preuve à maintes reprises.

Définition 25 (Graphe d'interaction)**.** *Soit F un réseau d'automates. Le* graphe d'interaction *de F est le graphe orienté qui est défini sur les sommets S, et qui possède une arête de s vers r si et seulement si s influence r. Si s influence positivement r, alors l'arête (s,r) est étiquetée par "+". Si s influence négativement r, alors l'arête (s,r) est étiquetée par "-".*

Les exemples 28 et 29 sont représentés sous la forme de graphes d'interaction dans les figures 4.1 et 4.2 respectivement.

Lorsque l'on peut représenter les interactions d'un réseau d'automates booléens par le biais d'arêtes positives et négatives, il devient intéressant de considerer le signe des cycles qui constituent le réseau. Le signe d'un cycle est simplement le produit des signes des arêtes qui le composent. L'étude de tels graphes d'interaction sans

nécessairement disposer des réseaux d'automates qui les définissent est un sujet de recherche actif (PAULEVÉ et RICHARD 2012 ; GADOULEAU et RICHARD 2016 ; ARACENA, RICHARD et SALINAS 2017 ; RICHARD 2018).

Définition 26 (Cycle signé)**.** *Soient F un réseau d'automates booléens et G son graphe d'interaction tels que les arêtes de G sont signées. Soit c un cycle de G et k le nombre d'arêtes au signe négatif qui le composent. Le cycle c est dit* positif *si k est pair, et* négatif *sinon.*

Exemple 31. *Considérons le réseau d'automates F' développé dans l'exemple 29, et son graphe d'interaction représenté en figure 4.2. Le cycle (d, e, g) dispose de deux arêtes négatives, (d, e) et (g, d). Ce nombre étant pair, le cycle (d, e, g) est donc positif.*

4.4 Mises à jour et exécutions

Un réseau d'automates est par nature un objet dynamique. Étant donné une configuration, la complexité du modèle apparaît lorsque l'on étudie les mises à jour successives du réseau sur cette configuration. Il n'existe cependant pas qu'une seule façon de mettre à jour un réseau ; en effet, de par sa nature décentralisée, chaque automate dans le réseau peut être considéré comme un acteur indépendant des autres, et l'on pourrait vouloir mettre à jour certains de ces acteurs sans mettre à jour les autres. Un automate seul est mis à jour en employant sa fonction locale ; comment exprimer la mise à jour du réseau dans sa totalité ?

Le protocole que l'on suit pour savoir quel automate mettre à jour à quel moment est appellé un *mode de mise à jour*. Dans le cadre de ce mémoire, nous définissons ces modes comme des séquences de mise à jour successives. Une mise à jour est un sous-ensemble des automates du réseau, de façon à ce que lorsqu'un automate s est inclus dans une mise à jour, cela signifie que cet automate exécutera sa fonction locale lors de cette mise à jour.

Définition 27 (Mise à jour)**.** *Soit F un réseau d'automates. Une* mise à jour *de F est un sous-ensemble $\delta \subseteq S$.*

Ainsi, mettre à jour un réseau d'automates d'après une mise à jour $\delta = \{a\}$ signifira l'application de la fonction locale f_a seule. La mise à jour $\delta = S$ signifira l'application de toutes les fonctions locales simultanément. L'application d'une mise à jour est une notion intuitive qui est définie comme suit.

Définition 28 (Application de mise à jour)**.** *Soit F un réseau d'automates, x une configuration et δ une mise à jour. L'*application de la mise à jour δ *sur la configuration x dans F est une nouvelle configuration notée $F_\delta(x)$, qui est définie par :*

$$\forall s \in S, F_\delta(x)_s = \begin{cases} f_s(x) & si\ s \in \delta \\ x_s & sinon \end{cases} .$$

L'application de la mise à jour $F_\delta(x)$ pour $\delta = S$ est également appellée la mise à jour *parallèle* du réseau d'automates F. Par confort nous la noterons $F(x)$.

Exemple 32. *Considérons le réseau d'automates F développé dans l'exemple 28, ainsi que trois mises à jour $\delta_1 = \{a\}$, $\delta_2 = \{b, c\}$ et S. Nous observons les faits suivants :*

$$\begin{aligned} F_{\delta_1}(000) &= 100 \\ F_{\delta_2}(000) &= 001 \\ F_S(000) &= 101 \\ F_{\delta_1}(101) &= 001 \\ F_{\delta_2}(101) &= 110 \\ F_S(101) &= 010 \end{aligned}$$

Il est naturel de vouloir mettre à jour un réseau d'automates plus d'une fois à la suite. Nous appellons une séquence de mises à jour un *mode de mise à jour*.

Définition 29 (Mode de mise à jour)**.** *Un* mode de mise à jour *est une séquence $\Delta = (\delta_1, \delta_2, \ldots)$, où pour tout k, δ_k est une mise à jour.*

Un tel mode de mise à jour peut être fini ou infini. Ces modes de mise à jour peuvent être classifiés selon certaines propriétés.

Définition 30 (Mode de mise à jour équitable)**.** *Soit Δ un mode de mise à jour. On appelle Δ un mode de mise à jour* équitable *si, dans le cas où Δ est fini, tout automate est inclus dans au moins une mise à jour de Δ, et si Δ est infini, chaque automate est inclus dans un nombre infini de mises à jour de Δ.*

Définition 31 (Mode de mise à jour parallèle)**.** *Le mode de mise à jour* parallèle*, noté Δ_P, est défini par $\Delta_P = (\delta)$ avec $\delta = S$.*

Le mode de mise à jour parallèle est alternativement appellé le mode de mise à jour *synchrone*.

Définition 32 (Mode de mise à jour séquentiel)**.** *Un mode de mise à jour Δ est dit* séquentiel *si et seulement si les mises à jour qui le composent sont des singletons et si leur ensemble est une partition de S.*

Définition 33 (Mode de mise à jour bloc-séquentiel)**.** *Un mode de mise à jour Δ est dit* bloc-séquentiel *si et seulement si l'ensemble des mises à jour qui le composent est une partition de S.*

On notera que, par définition, le mode de mise à jour parallèle et tout mode de mise à jour séquentiel sont également des modes de mise à jour blocs-séquentiels.

Exemple 33. *Soit $S = \{a, b, c\}$. Le mode de mise à jour parallèle est défini par $\Delta_P = (\{a, b, c\})$. Les modes de mise à jour $(\{a\}, \{b\}, \{c\})$, $(\{c\}, \{b\}, \{a\})$ et $(\{b\}, \{a\}, \{c\})$ sont des exemples de modes de mise à jour séquentiels, et $(\{b, c\}, \{a\})$ est un exemple de mode de mise à jour bloc-séquentiel.*

L'application d'un mode de mise à jour sur un réseau d'automates en partant d'une configuration donnée est appellée une *exécution* de ce réseau d'automates. Cette exécution est définie comme l'application successive des mises à jour qui composent le mode de mise à jour.

Définition 34 (Exécution)**.** *Soit F un réseau d'automates, $\Delta = (\delta_1, \delta_2, \ldots)$ un mode de mise à jour fini et x une configuration. On appelle exécution de F selon Δ sur x la configuration définie par les équations récursives suivantes :*

$$\begin{cases} F_{(\,)}(x) & = x \\ F_{(\delta_1, \delta_2, \ldots)}(x) & = F_{(\delta_2, \ldots)}(F_{\delta_1}(x)) \end{cases}.$$

Exemple 34. *Nous considérons F le réseau d'automates développé dans l'exemple 28. Soit $\Delta_1 = (\{a\}, \{b\}, \{c\})$ et $\Delta_2 = (\{b, c\}, \{a\})$. Nous observons les faits suivants :*

$$\begin{aligned} F_{\Delta_P \cdot \Delta_P \cdot \Delta_P}(000) &= 101 \\ F_{\Delta_1}(000) &= 110 \\ F_{\Delta_2}(000) &= 101 \\ F_{\Delta_P \cdot \Delta_P \cdot \Delta_P}(111) &= 010 \\ F_{\Delta_1}(111) &= 010 \\ F_{\Delta_2}(111) &= 010 \end{aligned}$$

L'exemple 34 illustre avec précision un fait connu dans la littérature des réseaux d'automates : le choix du mode de mise à jour fait varier drastiquement le calcul opéré par le réseau, et ce même en restreignant notre étude aux modes de mise à jour blocs-séquentiels voire même en comparant différent modes de mise à jour séquentiels (Robert 1986 ; Goles et Salinas 2008 ; Demongeot, Noual et Sené 2012 ; Aracena, Gómez et Salinas 2013 ; Noual et Sené 2018).

4.5 Graphe de la dynamique et attracteurs

Lorsque nous disposons d'un réseau d'automates et d'un certain mode de mise à jour, il est intéressant de représenter l'intégralité du calcul opéré par le réseau, en mettant en relation toutes les possibles configurations dudit réseau. Étant donné un mode de mise à jour Δ, le graphe de la dynamique d'un réseau d'automates est

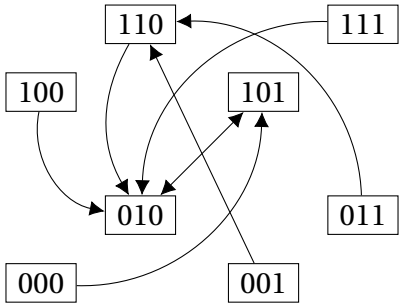

FIGURE 4.3 – Graphe de la dynamique parallèle du réseau d'automates développé dans l'exemple 28.

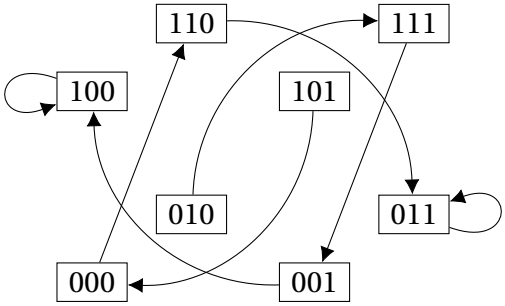

FIGURE 4.4 – Graphe de la dynamique parallèle du réseau d'automates développé dans l'exemple 29.

défini comme le graphe orienté dont les sommets sont les configurations du réseau et, pour deux configurations, le graphe disposera d'une arête orientée si et seulement si la première configuration, mise à jour sur le réseau avec Δ, donne la seconde configuration.

Définition 35 (Graphe de la dynamique)**.** *Soit F un réseau d'automates et Δ un mode de mise à jour. Le* graphe de la dynamique *de F selon Δ est le graphe orienté avec sommets dans Λ^S, tel que toute paire de configuration (x, y) est une arête du graphe si et seulement si $F_\Delta(x) = y$.*

La figure 4.3 représente le graphe de la dynamique selon le mode de mise à jour parallèle du réseau d'automates développé dans l'exemple 28.

La dynamique de modes de mise à jour fixés n'est pas le seul moyen d'étudier la dynamique d'un réseau d'automates. Il est en effet concevable de mettre à jour un réseau d'automates non pas par simple répétition d'un mode de mise à jour fini, mais par l'application non déterministe des fonctions locales des différent nœuds. Cette dynamique bien plus complexe peut également être représentée par ce qui est appellé un *graphe de la dynamique asynchrone*. Ce graphe possède une arête entre deux configurations étiquetée par un sommet s si et seulement si l'application de la

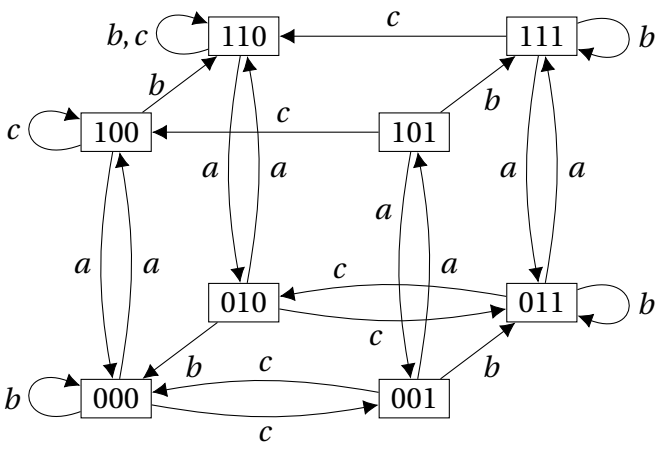

FIGURE 4.5 – Graphe de la dynamique asynchrone du réseau d'automates développé dans l'exemple 28.

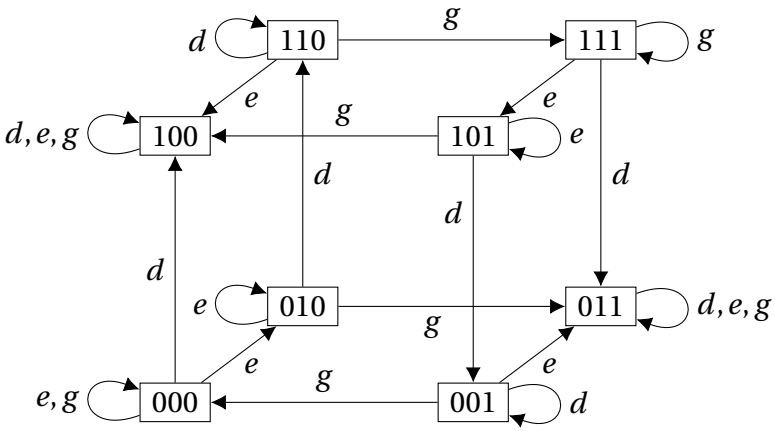

FIGURE 4.6 – Graphe de la dynamique asynchrone du réseau d'automates développé dans l'exemple 29.

fonction locale f_s seule fait passer de la première configuration à la seconde.

Définition 36 (Graphe de la dynamique asynchrone)**.** *Soit F un réseau d'automates. Le* graphe de la dynamique asynchrone *de F est le graphe orienté étiqueté avec sommets dans Λ^S et étiquettes dans S tel que (x, a, y) est une arête étiquetée du graphe si et seulement si $F_{\{a\}}(x) = y$.*

Les figures 4.5 et 4.6 représentent les graphes des dynamiques asynchrones des réseaux d'automates développés dans les exemples 28 et 29 respectivement.

Le graphe de la dynamique asynchrone représente toutes les façons de mettre à jour le réseau en utilisant des mises à jours successives qui ne concernent qu'un seul

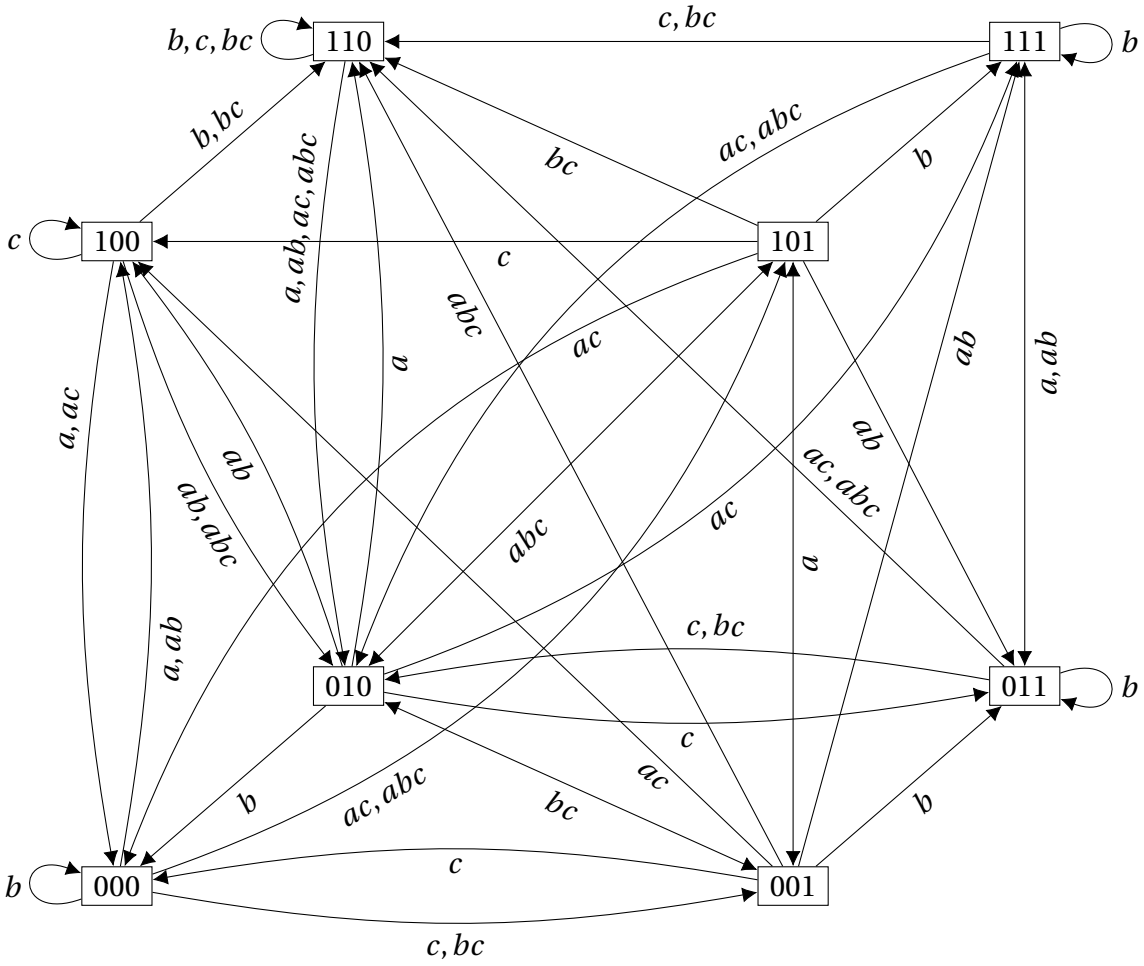

FIGURE 4.7 – Graphe de la dynamique totale du réseau d'automates développé dans l'exemple 28. L'étiquette ab représente la mise à jour $\delta = \{a, b\}$.

automate. Ainsi, ce graphe contient implicitement la dynamique de tous les modes de mise à jour sequentiels, entre autres.

Il est possible de représenter l'ensemble total de tous les modes de mises à jour sous la forme d'un graphe de la dynamique. Pour cela, il suffit d'étendre les étiquettes du graphe de la dynamique asynchrone à n'importe quel sous-ensemble de S. Ainsi, l'effet de n'importe quelle mise à jour depuis n'importe quelle configuration est répertorié. Nous appellons un tel graphe le *graphe de la dynamique totale*.

Définition 37 (Graphe de la dynamique totale)**.** *Soit F un réseau d'automates. Le* graphe de la dynamique totale *de F est le graphe orienté étiqueté avec sommets dans Λ^S et étiquettes dans $\wp(S) \setminus \emptyset$ tel que (x, δ, y) est une arête étiquetée du graphe si et seulement si $F_\delta(x) = y$.*

La figure 4.7 représente le graphe de la dynamique totale du réseau d'automates développé dans l'exemple 28.

Lorsque nous développons les détails de la dynamique d'un réseau d'automates, la complexité et la densité de l'information obtenue augmente. Ainsi, pour un réseau d'automates booléens de taille n, si les graphes de la dynamique obtenus sont tous des graphes de 2^n sommets, le nombre d'étiquettes contenues dans le graphe varie entre 2^n, $n \times 2^n$ ou 2^{2n}, en fonction de si l'on considère la dynamique d'un mode de mise à jour fixé, la dynamique asynchrone ou la dynamique totale respectivement. Dans les applications de ces modèles, le développement complet de ces dynamiques est ainsi très coûteux et presque toujours déraisonnable. Dans le cas non booléen à k états, ces mêmes dimensions deviennent k^n, $n \times k^n$ et k^{2n}.

Cependant, dans la plupart des applications des réseaux d'automates booléens, une compréhension extensive de la dynamique d'un réseau n'est pas nécessaire. En effet, on s'intéresse le plus souvent aux ensembles de configurations que l'on nomme les *attracteurs*. Ces configurations font parties de cycles de tailles variées ou de boucles dans le graphe de la dynamique du réseau. Ainsi, par exemple, dans le graphe représenté en figure 4.3, les configurations 010 et 101 font toute deux parties du même cycle de taille deux. Un tel cycle est appelé un *cycle limite*. Lorsqu'un attracteur est de taille 1, on l'appelle un *point fixe*.

Définition 38 (Attracteur)**.** *Soit G un graphe de dynamique. Un* attracteur *de G est une composante fortement connexe terminale de G.*

Définition 39 (Cycle limite)**.** *Soit F un réseau d'automates et Δ un mode de mise à jour. Un* cycle limite *est une séquence de configurations distinctes $(x_1, x_2, \ldots, x_k)$ telle que $F_\Delta(x_i) = x_{i+1}$ pour tout $0 < i < k$, $F_\Delta(x_k) = x_1$ et $k > 1$.*

Exemple 35. *Considérons le réseau d'automates F développé dans l'exemple 28. Le graphe de la dynamique parallèle de F, illustré en figure 4.3, dispose d'un seul attracteur, un cycle limite de taille 2 composé des configurations 010 et 101.*

Définition 40 (Point fixe)**.** *Soit F un réseau d'automates et Δ un mode de mise à jour. Un* point fixe *est une configuration x telle que $F_\Delta(x) = x$.*

Exemple 36. *Considérons le réseau d'automates F' développé dans l'exemple 29. Le graphe de la dynamique parallèle de F', illustré en figure 4.4, dispose de deux attracteurs qui sont des points fixes, les configurations 100 et 011.*

Dans le cas plus général des graphes de dynamique asynchrone ou totale, il est rare d'observer des cycles limites aussi clairement définis. On préfère donc en général définir le comportement limite d'un réseau par le biais d'attracteur complexe. Un attracteur complexe est un ensemble de configurations dont il est impossible pour le réseau de sortir.

Définition 41 (Attracteur complexe)**.** *Soit G un graphe de dynamique non-déterministe. Un* attracteur complexe *de G est une composante fortement connexe terminale de G de cardinal supérieur à 1.*

Un attracteur complexe est donc tout attracteur qui n'est pas un point fixe. Cette définition en invite une autre, celle du bassin d'attraction d'un attracteur, qui est composé de toutes les configurations en dehors de l'attracteur qui convergent asymptotiquement dans cet attracteur. Cette définition s'applique aux attracteurs complexes comme aux points fixes et aux cycles limites.

Définition 42 (Bassin d'attraction)**.** *Soit $G = (S, A)$ un graphe de dynamique, et S' un attracteur de G. Le bassin d'attraction de S' est l'ensemble des configurations $B_{S'}$ tel que :*

- *Pour tout $x \in B_{S'}$, et x' tel que $(x, x') \in A$, alors $x' \in B_{S'}$ ou $x' \in S'$.*
- *Pour tout $x \in B_{S'}$, il existe $x' \in S'$ et un chemin de x vers x' dans G.*

Exemple 37. *Considérons le réseau d'automates F' développé dans l'exemple 29. Le graphe de la dynamique asynchrone de F', illustré en figure 4.6, dispose de deux attracteurs qui sont des points fixes, les configurations* 100 *et* 011*. Le bassin d'attraction de ces deux attracteurs est l'ensemble des configurations* $\{000, 001, 010, 101, 110, 111\}$.

4.6 Résultats fondamentaux

Pour illustrer la difficulté des questions qui lui sont associées, nous proposons une présentation des résultats les plus fondateurs qui constituent notre compréhension actuelle de la dynamique des réseaux d'automates.

Il existe des cas particuliers de réseaux dont les attracteurs sont particulièrement bien compris. C'est le cas des réseaux d'automates acycliques, définis comme les réseaux d'automates dont le graphe d'interaction est acyclique. Ces objets sont étudiés en plus de détail au chapitre 6. Leur structure simple a permis la caractérisation suivante.

Théorème 1 (Robert 1980)**.** *Soit F un réseau d'automates. Si le graphe d'interaction de F est acyclique, alors sa dynamique totale n'a qu'un seul attracteur qui est un point fixe.*

Ce théorème démontre que les cycles dans le graphe d'interaction d'un réseau sont une condition nécessaire à la complexité de sa dynamique. Si un réseau est acyclique, alors toute configuration de ce réseau sous tout mode de mise à jour équitable converge vers un unique point fixe.

Les points fixes sont un sujet d'étude approfondi et s'avèrent particulièrement résilients au changement de mode de mise à jour. Le résultat suivant énonce que les points fixes d'un réseau d'automates sont les mêmes sous tout mode de mise à jour bloc-séquentiel. Cela se comprend intuitivement par le fait que, dans un point fixe,

aucun automate ne peut changer d'état, et ce, peu importe si ces automates sont mis à jour séparément ou en bloc.

Théorème 2 (ROBERT 1986)**.** *L'ensemble des points fixes d'un réseau d'automate est le même sous tout mode de mise à jour bloc-séquentiel.*

Ce théorème peut être étendu car un point fixe d'un réseau d'automates sous un mode de mise à jour bloc-séquentiel reste un point fixe sous toute itération du réseau, c'est à dire sous l'application de toute mise à jour.

Théorème 3 (Folklore)**.** *Soit F un réseau d'automate. Pour toute mise à jour δ, l'ensemble des points fixes de F sous un mode de mise à jour bloc-séquentiel est inclus dans l'ensemble des points fixes de F sous δ. La réciproque n'est pas vraie.*

La réciproque de ce théorème n'est pas vraie car certaines itérations peuvent provoquer l'apparition de points fixes par l'absence de la mise à jour d'un automate donné ou encore la répétition de la mise à jour d'un ou plusieurs automates au sein d'une période de mises à jour. Par exemple, le cas trivial de la mise à jour vide $\delta = \emptyset$ admet toute configuration comme point fixe, ce qui n'est évidemment pas le cas pour tout réseau d'automates mis à jour bloc-séquentiellement.

Ces premiers théorèmes sont simples et montrent que les points fixes de la dynamique d'un réseau d'automates sont résilients à la variation du mode de mise à jour. Reste cependant à comprendre les conditions spécifiques qui permettent l'observation d'un point fixe ou d'un attracteur complexe. Comme les cycles sont observés comme nécessaires à la complexité de la dynamique, il est pertinent de s'intéresser aux effets sur la dynamiques de la présence d'un cycle positif ou d'un cycle négatif dans le graphe d'interaction du réseau. Cette approche est représentée par les conjectures de Thomas, qui sont exprimées dans le contexte d'un mode de mise à jour asynchrone :

Conjecture 2 (THOMAS 1981)**.** *La présence d'un cycle positif dans le graphe d'interaction G d'un réseau F est nécessaire pour que F admette au moins 2 points fixes.*

Conjecture 3 (THOMAS 1981)**.** *La présence d'un cycle négatif dans le graphe d'interaction G d'un réseau F est nécessaire pour que F admette un attracteur complexe.*

Ces conjectures ont été prouvées dans le cadre booléen (REMY, MOSSÉ, CHAOUIYA et al. 2003), puis dans le cadre multivalué (RICHARD et COMET 2007 ; RICHARD 2010). La première conjecture, qui touche à la présence d'un cycle positif, s'avère vraie pour tout mode de mise à jour (NOUAL 2012 ; SENÉ 2012). En revanche, la conjecture qui touche à la présence d'un cycle négatif ne se généralise pas de cette façon car, par exemple, un circuit positif (un réseau d'automates composé seulement d'un cycle positif) admet des cycles limites sous le mode de mise à jour parallèle.

Ces conjectures et les théorèmes qui en découlent permettent d'exprimer deux intuitions : d'abord, les cycles positifs dans le graphe d'interaction d'un réseau sont nécessaires pour que ce réseau puisse se stabiliser de multiples façons ; les cycles positifs, en un sens, permettent une boucle de rétroaction positive qui font d'un réseau

un objet complexe mais stable. Ensuite, les cycles négatifs dans le graphe d'interaction d'un réseau sont nécessaires à l'apparition de comportements complexes instables. Ces résultats permettent également de souligner le rapport complexe entre mode de mise à jour et dynamique. Si les résultats qui portent sur les points fixes peuvent, par application des théorèmes 2 et 3, être généralisés à de nombreux modes de mise à jour, la caractérisation des cycles limites échappent à cette application.

La variation de la dynamique d'un réseau d'automates en fonction du mode de mise à jour étudié est un sujet actif du domaine. Par exemple, (GOLES et SALINAS 2008) établissent certaines différences significatives entre le mode de mise à jour parallèle et les modes de mise à jour séquentiels, en particulier dans leur application sur des réseaux structurés en couches. Similairement, (ARACENA, GOLES, MOREIRA et al. 2009) étudient certains critères qui permettent l'équivalence ou la différenciation de différents modes de mises à jours, et (NOUAL et SENÉ 2018) établissent un résultat qui touche à la sensibilité de la dynamique au synchronisme. Ces résultats s'appliquent à des classes spécifiques de réseaux d'automates. L'étude de l'effet du mode de mise à jour sur la dynamique est approfondi par (ARACENA, GOLES, MOREIRA et al. 2009 ; ARACENA, GÓMEZ et SALINAS 2013) qui proposent le formalisme de graphes de mise à jour, qui décrit le rapport entre le mode de mise à jour et le réseau d'automates, et permet de décrire des classes d'équivalence de modes de mises à jour. Certains problèmes de complexités reliés à ces graphes de mise à jour ont été caractérisés (ARACENA, FANCHON, MONTALVA et al. 2011 ; NOÛS, PERROT, SENÉ et al. 2020), tels que le problème de vérifier qu'un graphe donné est un graphe de mise à jour valide (qui est NP-complet), ou le problème de compter l'ensemble des étiquetages valide d'un graphe de mise à jour (qui est #P-complet).

Les travaux qui touchent au rapport entre variation du mode de mise à jour et variation de la dynamique expriment, dans leur ensemble, des solutions et des résultats qui s'appliquent à des cas particuliers et souvent spécifiques aux études concernées. Cela est attendu lors de l'étude d'une question aussi difficile, où l'absence de réponse générale et limpide fait converger la recherche vers des solutions plus simples dans des cas particuliers. Cependant, les travaux tels que (ARACENA, GÓMEZ et SALINAS 2013) nous montrent qu'il est encore possible de trouver des outils (ici les graphes de mise à jour) permettant d'aller plus loin dans la compréhension de l'influence de l'ordonnancement des mises à jour dans le temps.

Enfin, une portion de la littérature des réseaux d'automates pose la question du comptage de leurs attracteurs. Ces travaux établissent des bornes supérieures (ARACENA 2008 ; RICHARD 2009) ou comptent directement les attracteurs de classes particulières de réseaux d'automates (DEMONGEOT, NOUAL et SENÉ 2012 ; ARACENA, RICHARD et SALINAS 2017). Bien que l'approche de ces travaux constitue l'expression et la résolution partielle d'un simple problème de comptage, il peut être dit que l'expression la plus générale de ce problème de comptage – déterminer le nombre maximal d'attracteurs admis par un réseau d'automates – ouvre la voie vers une version discrète du second volet du seixième problème de Hilbert (ILYASHENKO 2002). La question générale du comptage des attracteurs de tout réseau d'automates s'avère une

question d'une redoutable difficulté, malgré sa résolution dans des cas de structures simples comme les réseaux d'automates acycliques, ou les circuits (c'est-à-dire les réseaux uniquement composés d'un cycle).

5 Modules

5.1 Introduction

La prédiction de la dynamique d'un réseau d'automates est un problème difficile, de par le fait que cette dynamique est exponentielle en la taille du réseau. Une des approches pour contourner ce problème est de diviser le réseau étudié en plusieurs sous-réseaux, dans l'espoir que la prédiction des éléments séparés permettent la prédiction de l'ensemble.

Cette approche modulaire n'est pas nouvelle et a été explorée de plusieurs façons dans la littérature. Certains travaux (MILO, SHEN-ORR, ITZKOVITZ et al. 2002 ; U. ALON 2003) ont rassemblés des réseaux d'influences tirés de la biologie, de la biochimie, de l'écologie et de l'ingénierie et étudient la réccurence de motifs, c'est-à-dire des sous-graphes spécifiques dont la présence est anormalement haute dans ces réseaux naturels comparés à des réseaux générés aléatoirement. D'autres travaux (BERNOT et TAHI 2009 ; SIEBERT 2009 ; DELAPLACE, KLAUDEL, MELLITI et al. 2012) étudient des formalismes qui permettent la définition de sous-réseaux dans un réseau d'automate, avec l'objectif de comprendre la dynamique d'un réseau plus large à partir de l'étude de parties de ce réseau. Ces travaux développent également des méthodes qui permettent cette division en parties d'une façon efficace et pertinente. Un travail récent (ALKHUDHAYR et STEGGLES 2019) introduit une notion de composition de réseau d'automates booléens qui repose sur la fusion de deux automates. La fonction locale de l'automate obtenu par cette fusion est définie comme la composition par un opérateur booléen des fonctions locales des automates fusionnés.

Notre approche peut être rapprochée de travaux dans le cadre non discret des réseaux de réactions (BAEZ et POLLARD 2017), où des entrées sont rajoutées à ces systèmes qui modélisent des réactions chimiques par l'emploi de réseaux de Petri ouverts.

Dans le cadre des réseaux d'automates, notre formalisme diffère des précédents par le fait qu'il ne vise pas à décrire la nature d'un sous-réseau, mais bien la possibilité de tout réseau d'être influencé par un réseau extérieur par le biais d'entrées que nous rajoutons au modèle. Ainsi, la propriété de faire partie d'un réseau plus large peut être comprise comme la réduction de l'évolution de ces entrées. Nous appellons ces réseaux disposant d'entrées les *modules*. Ce chapitre présente les définitions associées aux modules en section 5.2, et présente un ensemble de résultats les concernant en section 5.3.

5.2 Définitions

5.2.1 Entrées, fonctions locales, modules et graphe d'interaction

Les modules sont une généralisation des réseaux d'automates. En ces termes, la plupart des définitions développées pour les réseaux d'automates s'appliquent également aux modules. Ainsi, si un réseau d'automates est défini avec un ensemble d'automates S ainsi qu'un ensemble d'états Λ, un module a également un ensemble d'*entrées*, que nous appellons par convention I . Par convention de nouveau, les entrées contenues dans cet ensemble sont désignées par des lettres grecques, en commencant par α.

Les entrées d'un module représentent des influences extérieures, qui possèdent une valeur dans Λ à tout instant, et qui peuvent influencer les fonctions locales du module. Pour pouvoir définir cette influence, il est nécessaire de définir un état pour ces entrées. Cela est fait par le biais d'une configuration d'entrée.

Définition 43 (Configuration d'entrée)**.** *Soit I un ensemble d'entrées et Λ un ensemble d'états. Une* configuration d'entrée *est un vecteur défini sur Λ^I.*

Par convention, nous noterons les configurations d'entrée par la lettre i.

Concaténer une configuration d'entrée i avec une configuration standard x nous donne un vecteur $x \cdot i$ défini sur $\Lambda^{S \cup I}$. Ce vecteur contient toute l'information nécessaire pour mettre à jour les fonctions locales du module. Ainsi, nous pouvons désormais développer la définition des fonctions locales des modules.

Définition 44 (Fonction locale d'un module)**.** *Soit S un ensemble d'automates, I un ensemble d'entrées et Λ un ensemble d'états. Une* fonction locale *d'un module est une fonction f définie sur $\Lambda^{S \cup I} \to \Lambda$.*

Nous pouvons à présent définir un module par une définition similaire à celle d'un réseau d'automates.

Définition 45 (Module)**.** *Soit S un ensemble d'automates, I un ensemble d'entrées et Λ un ensemble d'états. Un* module *associe à tout $s \in S$ une fonction locale $f_s : \Lambda^{S \cup I} \to \Lambda$.*

En ces termes, les entrées d'un module se comportent comme des automates dans le sens où leur valeur est définie dans une configuration et utilisée pour influencer les automates du réseau. Cependant, les entrées diffèrent des automates dans le fait qu'elles ne possèdent pas elles-mêmes de fonctions locales. Les entrées sont conçues comme une influence extérieure au réseau, et leur valeur est arbitrairement définie à tout instant.

Exemple 38. *Soit $S = \{a, b, c\}$, $I = \{\alpha\}$ et $\Lambda = \{0, 1\}$. Soit M le module qui à a, b et c associe les fonctions suivantes : $f_a(x \cdot i) = \neg x_a \wedge i_\alpha$, $f_b(x \cdot i) = x_a \vee x_c$, et $f_c(x \cdot i) = (\neg x_a \wedge \neg x_c) \vee i_\alpha$.*

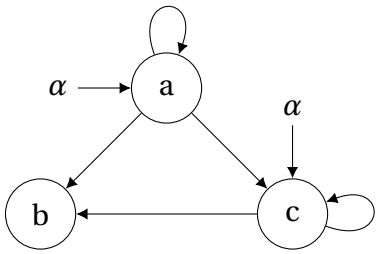

FIGURE 5.1 – Graphe d'interaction du module développé dans l'exemple 38.

L'influence qu'une entrée porte sur un automate se définit de façon similaire à l'influence entre deux automates.

Définition 46 (Influence d'une entrée)**.** *Soit M un module, α une entrée de I et s un automate de M. On dit que α* influence *s si et seulement s'il existe une configuration x et une paire de configurations d'entrée i et i' telles que $i_\beta = i'_\beta \iff \beta \neq \alpha$, et $f_s(x \cdot i) \neq f_s(x \cdot i')$.*

Tout comme les réseaux d'automates, les modules sont propices à la représentation sous la forme d'un graphe. Le *graphe d'interaction* d'un module se définit de façon pratiquement équivalente au graphe d'interaction d'un réseau d'automates, à l'ajout prêt de flèches qui représentent l'influence des entrées du module sur les automates.

Définition 47 (Graphe d'interaction de module)**.** *Soit M un module. On définit le* graphe d'interaction *de M comme le graphe orienté qui est défini sur les sommets de S, qui possède une arête de s vers r si et seulement si s influence r. De plus, chaque sommet s est décoré d'une flèche entrante annotée α pour chaque entrée α qui influence s.*

L'exemple 38 est représenté sous la forme d'un graphe d'interaction dans la figure 5.1.

5.2.2 Mises à jour, exécutions et graphe de la dynamique

Pour x une configuration et i une configuration d'entrée, un module M peut être mis à jour de la même façon qu'un réseau d'automates ; par le biais d'une sélection de fonctions locales de M qui s'appliquent simultanément pour altérer la configuration x. Cette sélection est représentée par une mise à jour δ, dont la définition ne change pas ; il s'agit toujours d'un sous-ensemble de S. La définition de l'application d'une telle mise à jour sur un module est intuitive.

Définition 48 (Application de mise à jour sur un module)**.** *Soit M un réseau d'automates, x une configuration, i une configuration d'entrée et δ une mise à jour. L'*application de la mise à jour *δ sur les configurations x et i dans M est une nouvelle configuration notée $M_\delta(x \cdot i)$, qui est définie par :*

$$\forall s \in S, M_\delta(x \cdot i)_s = \begin{cases} f_s(x \cdot i) & si\ s \in \delta \\ x_s & sinon \end{cases} .$$

Exemple 39. *Considérons le module M développé dans l'exemple 38, ainsi que trois mises à jour $\delta_1 = \{a\}$, $\delta_2 = \{b, c\}$ et S. Nous observons les faits suivants :*

$$\begin{aligned} M_{\delta_1}(000 \cdot 0) &= 000 \\ M_{\delta_2}(000 \cdot 0) &= 001 \\ M_S(000 \cdot 0) &= 001 \\ M_{\delta_1}(101 \cdot 1) &= 001 \\ M_{\delta_2}(101 \cdot 1) &= 011 \\ M_S(101 \cdot 1) &= 011 \end{aligned}$$

On note que l'application d'une mise à jour sur un module ne génère pas une nouvelle configuration pour les entrées de ce module. Les configurations d'entrée sont considérées indépendantes du fonctionnement interne du module. Cela signifie que sur une séquence de mises à jour, l'affectation d'états aux entrées du module pourra évoluer de façon complètement arbitraire. Pour refléter cette nature arbitraire, nous définissons la notion de séquence d'entrée.

Définition 49 (Séquence d'entrée)**.** *Soit I un ensemble d'entrée. Une* séquence d'entrée *est une séquence $J = (i_1, i_2, \ldots)$ finie ou infinie, où chaque i_k est une configuration d'entrée définie sur I.*

Ainsi, tout ce qui est nécessaire pour mettre à jour un module sur plus d'une mise à jour à la suite, est une configuration initiale x, un mode de mise à jour Δ, et une séquence d'entrée J au moins aussi longue que Δ.

Définition 50 (Exécution de module)**.** *Soit M un module, x une configuration, $\Delta = (\delta_1, \delta_2, \ldots)$ un mode de mise à jour fini et $J = (i_1, i_2, \ldots)$ une séquence d'entrée au moins aussi longue que Δ. On appelle* exécution *de M selon Δ sur x et J la configuration définie par les équations récursives suivantes :*

$$\begin{aligned} M_{(\delta_1, \delta_2, \ldots)}(x, (i_1, i_2, \ldots)) &= M_{(\delta_2, \ldots)}(M_{\delta_1}(x \cdot i_1), (i_2, \ldots)) \\ M_{(\,)}(x, J) &= x \end{aligned}$$

Exemple 40. *Nous considérons M le module développé dans l'exemple 38. Soit $\Delta_1 = (\{a\}, \{b\}, \{c\})$, $\Delta_2 = (\{b, c\}, \{a\})$, et $J = (1, 0, 1)$. Nous observons les faits suivant :*

$$M_{\Delta_P \cdot \Delta_P \cdot \Delta_P}(000, J) = 101$$

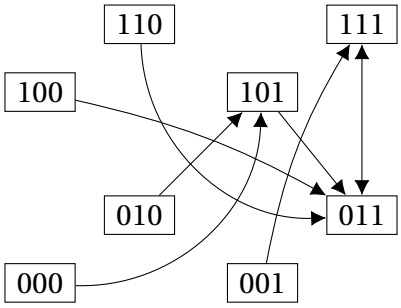

FIGURE 5.2 – Graphe de la dynamique parallèle avec séquence d'entrée $J = (1)$ du module développé dans l'exemple 38.

$$M_{\Delta_1}(000, J) = 011$$
$$M_{\Delta_2}(000, J) = 001$$
$$M_{\Delta_P \cdot \Delta_P \cdot \Delta_P}(111, J) = 111$$
$$M_{\Delta_1}(111, J) = 011$$
$$M_{\Delta_2}(111, J) = 011$$

Nous étendons la définition de graphe de la dynamique d'un réseau d'automates aux modules. Pour ce faire, il nous faut rajouter une information supplémentaire : en plus du mode de mise à jour Δ, l'on doit fournir une séquence d'entrée J de taille suffisament longue pour définir une exécution. On définit ensuite intuitivement le graphe de la dynamique en exécutant le module sur ces paramètres sur toutes les configurations possibles, et en représentant le résultat sous la forme d'un graphe.

Définition 51 (Graphe de la dynamique d'un module)**.** *Soit M un module, Δ un mode de mise à jour et J une séquence d'entrée au moins aussi longue que Δ. Le graphe de la dynamique de M selon Δ et J est le graphe orienté avec sommets dans Λ^S, tel que toute paire de configuration (x, y) est une arête du graphe si et seulement si $M_\Delta(x, J) = y$.*

La figure 5.2 représente le graphe de la dynamique selon le mode de mise à jour parallèle et la séquence d'entrée $J = (1)$ du module développé dans l'exemple 38.

Une telle représentation souffre d'un manque de variation sur la configuration d'entrée. Là où le graphe de la dynamique d'un mode de mise à jour donné d'un réseau d'automates était réduit à décrire uniquement le comportement pour un seul mode de mise à jour, le graphe de la dynamique d'un module est réduit à ne décrire que son comportement pour un mode de mise à jour et une séquence d'entrée. Ainsi, nous ne considérons pas cet objet comme une représentation fidèle des très nombreux comportements dont un module est capable.

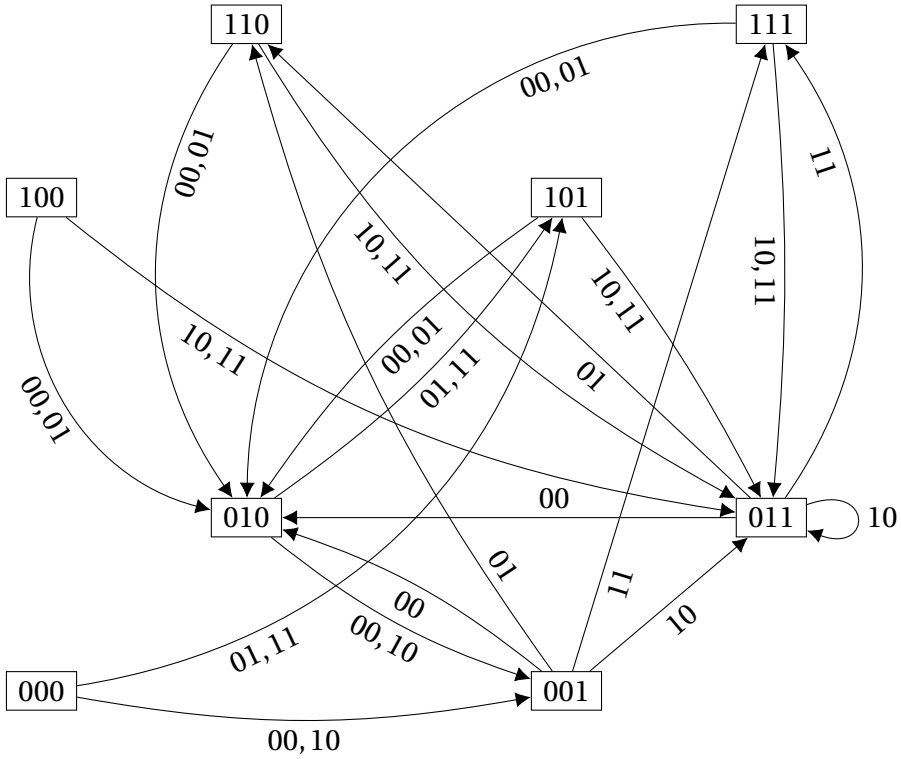

FIGURE 5.3 – Graphe de la dynamique d'entrée avec mode de mise à jour $\Delta = (\{b,c\},a)$ du module développé dans l'exemple 38. L'étiquette 00 représente la séquence d'entrée $J = (0,0)$.

Si la définition ci-dessus est une extension intuitive de la définition équivalente dans le cadre des réseaux d'automates, nous proposons une définition nouvelle propre au cas des modules. Ce que nous appellons graphe de la dynamique d'entrée est défini à partir d'un module et d'un mode de mise à jour, et représente toutes les exécutions possibles pour toutes les séquences d'entrée possibles.

Définition 52 (Graphe de la dynamique d'entrée)**.** *Soit M un module et Δ un mode de mise à jour. Le* graphe de la dynamique d'entrée *de M selon Δ est le graphe orienté et étiqueté avec sommets dans Λ^S et dont l'ensemble d'étiquetage est l'ensemble des séquences d'entrée de longueur égale à la taille de Δ, tel que le tuple (x,J,y) est une arête du graphe si et seulement si $M_\Delta(x,J) = y$.*

La figure 5.3 représente le graphe de la dynamique d'entrée selon le mode de mise à jour $\Delta = (\{b,c\},\{a\})$ du module développé dans l'exemple 38.

En plus des graphes de la dynamique donnés par des modes de mises à jours fixés, nous considérons l'équivalent du graphe de la dynamique asynchrone dans le contexte des modules. Ce graphe qui, dans le contexte des réseaux d'automates renseigne l'effet de toutes les mises à jours d'un seul automate à partir de toutes les configurations, doit

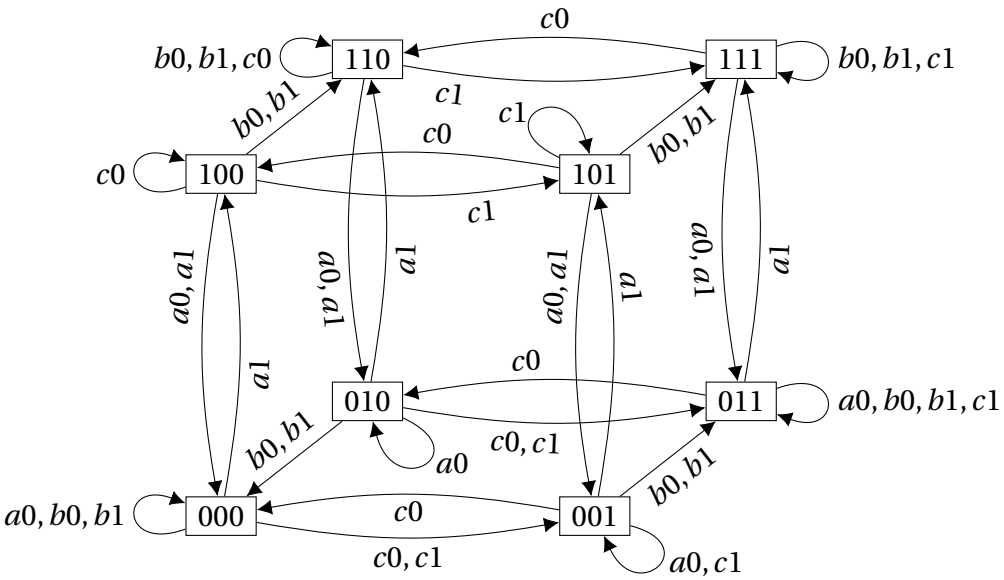

FIGURE 5.4 – Graphe de la dynamique asynchrone du module développé dans l'exemple 38. L'étiquette $a0$ signifie la mise à jour du sommet a avec l'entrée α évaluée à 0.

ici être augmenté de l'information concernant l'affectation des entrées au moment de la mise à jour. Cela signifie que l'ensemble des étiquettes sur les arêtes sortantes à chaque nœud sera le produit cartésien de l'ensemble des automates et de l'ensemble des configurations d'entrée.

Définition 53 (Graphe de la dynamique asynchrone d'un module)**.** *Soit M un module. Le* graphe de la dynamique asynchrone *de M est le graphe orienté étiqueté avec sommets dans Λ^S et étiquettes dans $S \times \Lambda^I$ tel que $(x,(a,i),y)$ est une arête étiquetée du graphe si et seulement si $F_{\{a\}}(x \cdot i) = y$.*

La figure 5.4 représente le graphe de la dynamique asynchrone du module développé dans l'exemple 38.

Ainsi, le graphe de la dynamique asynchrone d'un module permet de décrire le comportement de tout mode de mise à jour composé de singletons, et ce quelle que soit la séquence d'entrée utilisée. Afin de pouvoir étendre cette représentation à tout mode de mise à jour, nous devons introduire la notion correspondante du *graphe de la dynamique totale,* qui ici devra contenir l'information de la configuration d'entrée en addition à l'information de la mise à jour considérée.

Définition 54 (Graphe de la dynamique totale d'un module)**.** *Soit M un module. Le* graphe de la dynamique totale *de M est le graphe orienté étiqueté avec sommets dans Λ^S et étiquettes dans $\wp(S) \times \Lambda^I$ tel que $(x,(\delta,i),y)$ est une arête étiquetée du graphe si et seulement si $M_\delta(x \cdot i) = y$.*

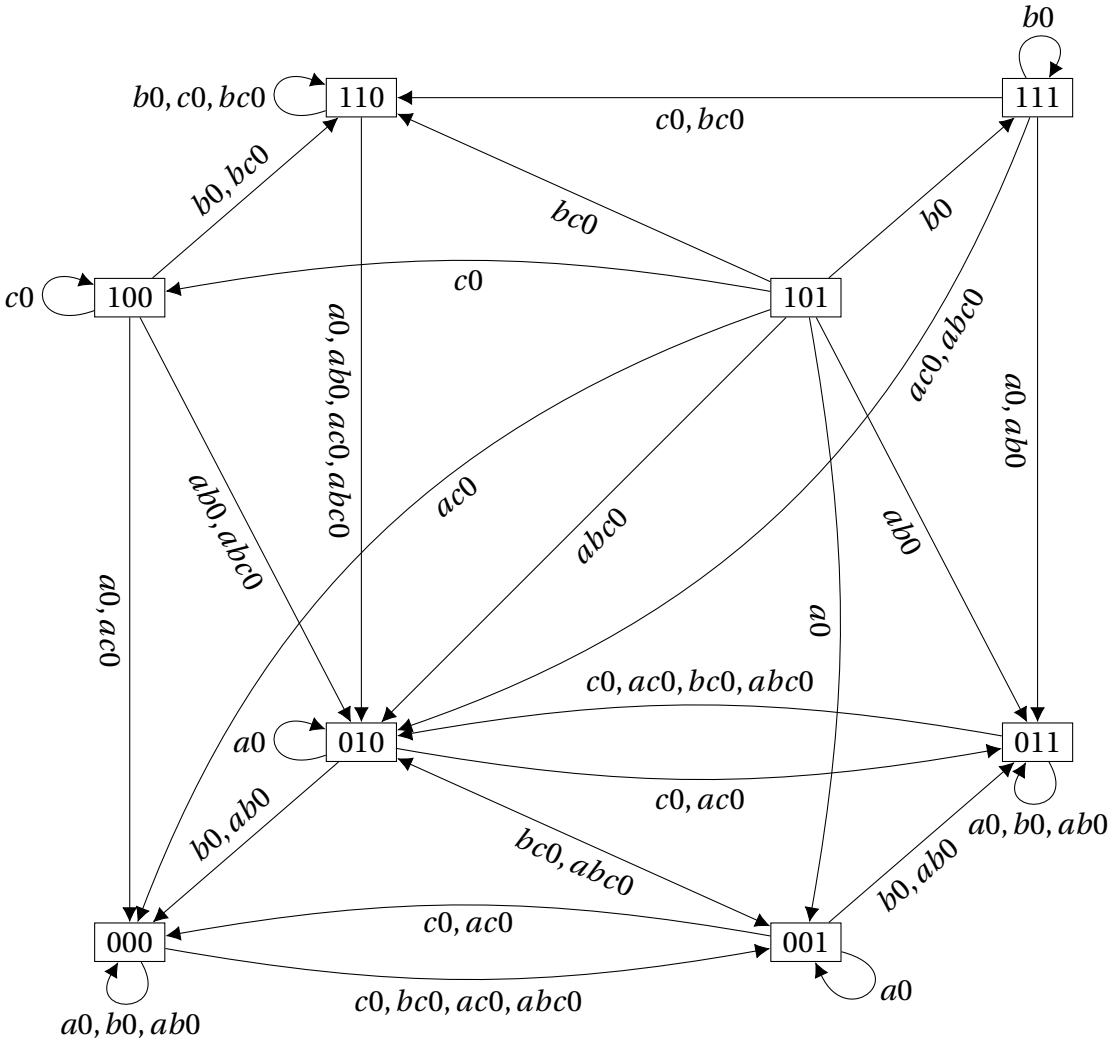

FIGURE 5.5 – Graphe de la dynamique totale du module développé dans l'exemple 38, réduit aux transitions avec la configuration d'entrée $i_\alpha = 0$. L'étiquette $ab0$ représente la mise à jour $\delta = \{a, b\}$ associée à la configuration d'entrée qui associe 0 à α.

Les figures 5.5 et 5.6 représentent le graphe de la dynamique totale du module développé dans l'exemple 38.

Nous faisons le constat assez prévisible que la taille de ces graphes est d'autant plus grande que le nombre d'entrées augmente. Plus spécifiquement, là où le nombre de sommets de chaque graphe est toujours le même, le nombre total d'arêtes étiquetées explose encore plus lourdement que dans le cas des réseaux d'automates. La taille de ces objets est détaillée dans la table en figure 5.7.

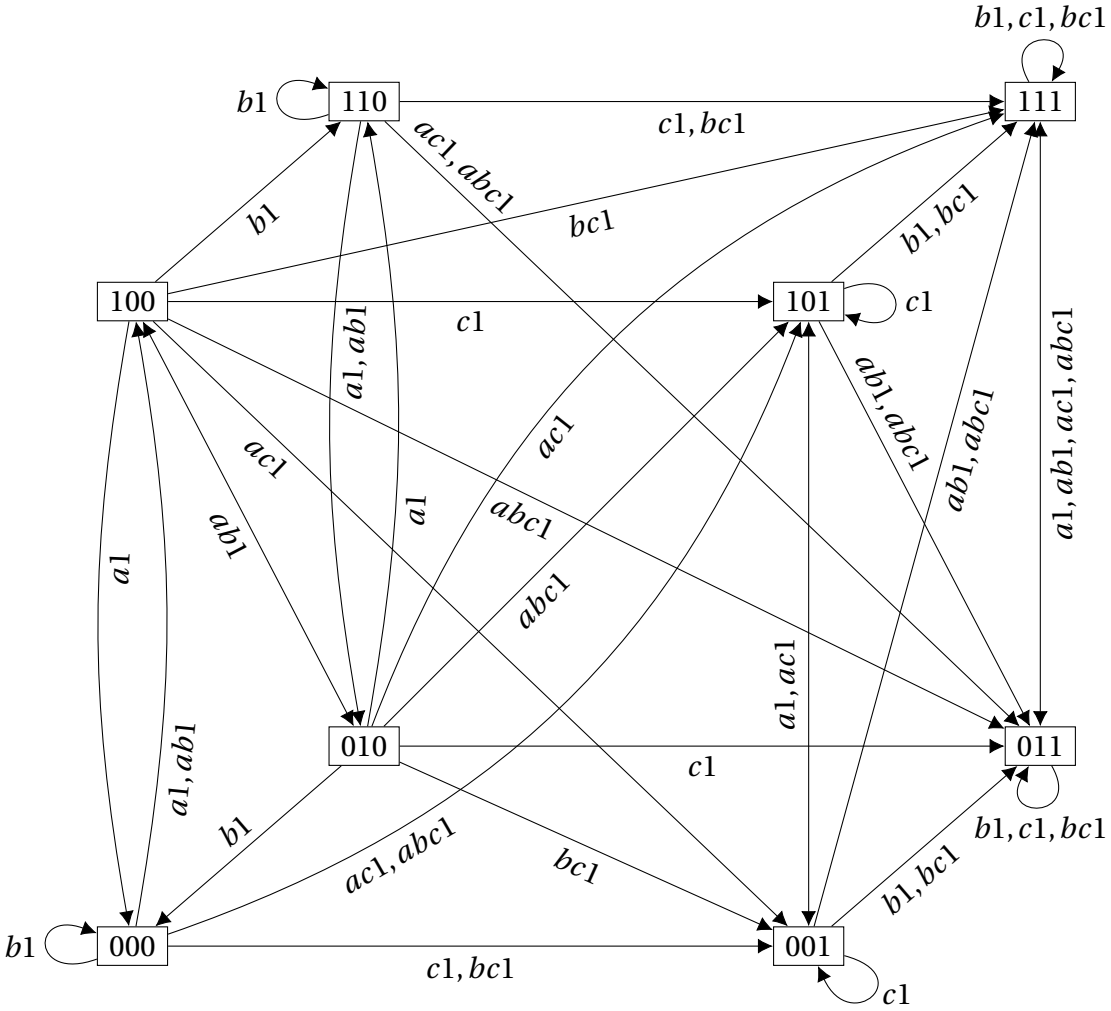

FIGURE 5.6 – Graphe de la dynamique totale du module développé dans l'exemple 38, réduit aux transitions avec la configuration d'entrée $i_\alpha = 1$. L'étiquette $ab1$ représente la mise à jour $\delta = \{a, b\}$ associée à la configuration d'entrée qui associe 1 à α.

5.2.3 Branchements

Les modules peuvent être transformés pour remplacer l'influence extérieure d'une entrée par l'influence spécifique d'un automate. Cette transformation est faite par le biais de branchements, qui sont de deux types : récursif, et non récursif. Ces deux types de modification opèrent par la substitution de variables dans les configurations d'une exécution. Dans le cas du branchement non récursif, les variables d'entrée retirées sont remplacées par les variables d'automates d'un autre réseau. Dans le cas du branchement récursif, les variables d'entrées retirées sont remplacées par les

Type de réseau	Module		Réseau d'automates	
Mesure	Nœuds	Arêtes	Nœuds	Arêtes
G. d'interaction	n	$n \times (n+i)$	n	n^2
G. de la dynamique	l^n	l^n	l^n	l^n
G. de la dynamique d'entrée	l^n	l^{nid}	N/A	N/A
G. de la dynamique asynchrone	l^n	$n \times l^{ni}$	l^n	$n \times l^n$
G. de la dynamique totale	l^n	$2^n \times l^{ni}$	l^n	$2^n \times l^n$

FIGURE 5.7 – Table renseignant la taille maximale des différents graphes de la dynamique explorés dans cette section, lorsqu'applicable. Les nombres l, n, i et d représentent respectivement le nombre d'états, le nombre d'automates, le nombre d'entrées et la longueur du mode de mise à jour appliqué.

variables d'automates du réseau lui-même.

Pour un ensemble de réseaux donné, il existe une multitude de façons différentes de combiner entrées et automates. Même en considérant le cas plus simple du branchement récursif d'un réseau seul, le nombre total de branchements possibles est de $i \times (n+1)$, où i est le nombre d'entrées, et n le nombre d'automates. C'est pour cette raison que nous définissons pour chaque branchement une fonction $\mathtt{w} : I \nrightarrow S$, où $\mathrm{dom}(\mathtt{w})$ est l'ensemble d'entrées du réseau influencé par le branchement, et $\mathrm{img}(\mathtt{w})$ est l'ensemble des automates du réseau dont l'influence va remplacer celle des entrées dans $\mathrm{dom}(\mathtt{w})$. Cette fonction, dite fonction de branchement, est une fonction partielle ; en effet, il est tout à fait possible de laisser des entrées non affectées par un branchement.

Nous rappellons que l'utilisation de l'opérateur de composition $\circ$ entre vecteurs et fonctions est définie en section 1.3.

Définition 55 (Branchement récursif)**.** *Soit M un module, et $\mathtt{w} : I \nrightarrow S$ une fonction de branchement. On définit le branchement récursif de M par $\mathtt{w}$, noté $\circlearrowright_{\mathtt{w}} M$, le module défini sur l'ensemble d'automates S et l'ensemble d'entrées $I \setminus dom(\mathtt{w})$ qui, pour tout automate s, définit la fonction locale f'_s telle que*

$$f'_s(x' \cdot i') = f_s((x' \cdot i') \circ \hat{\mathtt{w}})$$

où f_s est la fonction locale de s dans M, et $\hat{\mathtt{w}}$ est la fonction définie sur $S \cup I \to S \cup I \setminus dom(\mathtt{w})$, telle que

$$\hat{\mathtt{w}}(a) = \begin{cases} \mathtt{w}(a) & si\ a \in dom(\mathtt{w}) \\ a & sinon \end{cases}.$$

Ainsi, le résultat d'un branchement récursif est un nouveau module dans lequel, pour tout α dans le domaine de $\mathtt{w}$, l'entrée α est absente du module, et son emploi dans le réseau est remplacé par l'automate $\mathtt{w}(\alpha)$.

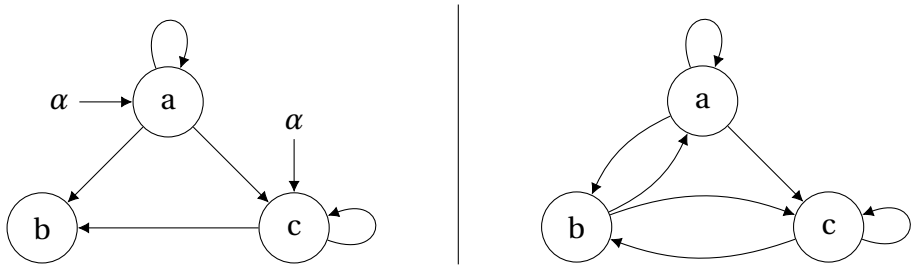

FIGURE 5.8 – Illustration du branchement décrit dans l'exemple 41. À gauche de la figure, le graphe d'interaction du module M. À droite de la figure, le graphe d'interaction du réseau d'automates $\circlearrowleft_{\mathtt{w}} M$.

Exemple 41. *Nous considérons le module M développé dans l'exemple 38. Soit $\mathtt{w}$ la fonction de branchement définie par $dom(\mathtt{w}) = \{b\}$ et $\mathtt{w}(\alpha) = b$. Le résultat du branchement récursif $\circlearrowleft_{\mathtt{w}} M$ est le réseau d'automate qui définit les fonctions locales*

$$f'_a(x \cdot i) = f_a((x \cdot i) \circ \hat{\mathtt{w}}) = \neg(x \cdot i)_{\hat{\mathtt{w}}(a)} \wedge (x \cdot i)_{\hat{\mathtt{w}}(\alpha)} = \neg(x \cdot i)_a \wedge (x \cdot i)_b = \neg x_a \wedge x_b$$
$$f'_b(x \cdot i) = f_b((x \cdot i) \circ \hat{\mathtt{w}}) = (x \cdot i)_{\hat{\mathtt{w}}(a)} \vee (x \cdot i)_{\hat{\mathtt{w}}(c)} = (x \cdot i)_a \vee (x \cdot i)_c = x_a \vee x_c$$
$$f'_c(x \cdot i) = f_c((x \cdot i) \circ \hat{\mathtt{w}}) = (\neg(x \cdot i)_{\hat{\mathtt{w}}(a)} \wedge \neg(x \cdot i)_{\hat{\mathtt{w}}(c)}) \vee (x \cdot i)_{\hat{\mathtt{w}}(\alpha)} = (\neg x_a \wedge \neg x_c) \vee x_b.$$

La figure 5.8 représente les graphes d'interaction des deux modules développés dans l'exemple 41.

L'exemple 41 fait la démonstration d'un phénomène propre à certains branchements récursifs. Lorsque la fonction de branchement $\mathtt{w}$ est une fonction totale, toutes les entrées sont remplacées par des automates et, par conséquent, le module obtenu ne dispose pas d'entrées. Dans ce cas, il est correct de désigner le module obtenu comme un réseau d'automates.

Là où le branchement récursif permet d'introduire de nouveaux cycles dans un module, le branchement non récursif permet d'introduire de nouveaux automates. Pour ce faire, deux modules différents sont combinés.

Définition 56 (Branchement non récursif)**.** *Soient M et M' deux modules tels que $S \cap S' = I \cap I' = \emptyset$, et $\mathtt{w}$ une fonction de branchement définie sur $I' \twoheadrightarrow S$. On définit le branchement non récursif de M sur M' d'après $\mathtt{w}$, noté $M \rightarrowtail_{\mathtt{w}} M'$, comme le module ayant pour ensemble d'automates $S \cup S'$ et pour ensemble d'entrées $I \cup I' \setminus dom(\mathtt{w})$ tel que, pour tout automate s, le module définit la fonction locale*

$$f''_s(x'' \cdot i'') = \begin{cases} f_s(x''|_S \cdot i''|_I) & si\ s \in S \\ f'_s((x'' \cdot i''|_{I'}) \circ \hat{\mathtt{w}}) & si\ s \in S' \end{cases}$$

où f_s et f'_s sont les fonctions locales de s dans M et M' respectivement, et $\hat{\mathtt{w}}$ est la fonction

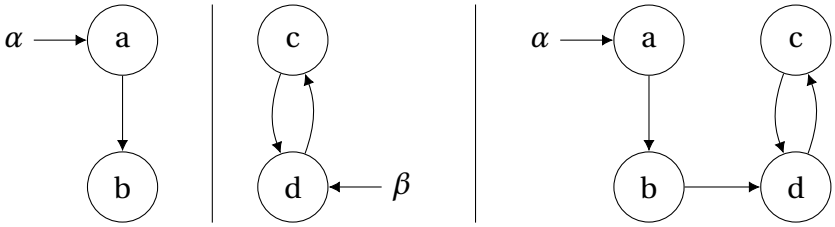

FIGURE 5.9 – Illustration du branchement décrit dans l'exemple 42. De gauche à droite dans la figure sont représentés les graphes d'interaction des modules M, M' et $M \rightarrowtail_{\mathtt{w}} M'$.

définie sur $S' \cup I' \rightarrow S \cup S' \cup I' \setminus dom(\mathtt{w})$ *telle que*

$$\hat{\mathtt{w}}(a) = \begin{cases} \mathtt{w}(a) & si\ a \in dom(\mathtt{w}) \\ a & sinon \end{cases} .$$

Exemple 42. *Soit M le module tel que $S = \{a, b\}$, $I = \{\alpha\}$, $f_a(x \cdot i) = i_\alpha$ et $f_b(x \cdot i) = \neg x_a$. Soit M' le module tel que $S' = \{c, d\}$, $I' = \{\beta\}$, $f_c(x \cdot i) = \neg x_d$ et $f_d(x \cdot i) = x_c \vee i_\beta$. Soit $\mathtt{w}$ la fonction de branchement définie sur $\{\beta\} \rightarrow S$ telle que $\mathtt{w}(\beta) = b$. Le module $M \rightarrowtail_{\mathtt{w}} M'$ est défini sur les ensembles $S'' = \{a, b, c, d\}$, $I'' = \{\alpha\}$ et définit les fonctions locales*

$$\begin{aligned} f''_a(x \cdot i) &= f_a(x|_S \cdot i|_I) = i_\alpha \\ f''_b(x \cdot i) &= f_b(x|_S \cdot i|_I) = \neg x_a \\ f''_c(x \cdot i) = f_c((x \cdot i) \circ \hat{\mathtt{w}}) &= \neg (x \cdot i)_{\hat{\mathtt{w}}(d)} = \neg (x \cdot i)_d = \neg x_d \\ f''_d(x \cdot i) = f_d((x \cdot i) \circ \hat{\mathtt{w}}) = (x \cdot i)_{\hat{\mathtt{w}}(c)} \vee (x \cdot i)_{\hat{\mathtt{w}}(\beta)} &= (x \cdot i)_c \vee (x \cdot i)_b = x_c \vee x_b. \end{aligned}$$

La figure 5.9 représente les graphes d'interaction des trois modules développés dans l'exemple 42.

Dans notre utilisation des outils que sont les branchement non récursifs, un cas de fonction de branchement ressort comme particulièrement utile. Nous parlons ici du cas où un branchement non récursif est appliqué avec une fonction de branchement dont le domaine est vide. Cette opération se contente de simplement juxtaposer deux modules sans altérer leurs comportements. Cette opération parait suffisament distincte et utile pour mériter une notation particulière, et nous l'utilisons pour définir l'union sur les modules.

Définition 57 (Union de modules)**.** *Soient M et M' deux modules. On définit l'*union *de ces modules, notée $M \cup M'$, comme le résultat de l'opération $M \rightarrowtail_{\mathtt{w}} M'$, pour $\mathtt{w}$ la fonction telle que $dom(\mathtt{w}) = \emptyset$.*

Similairement, on peut définir le cas particulier du branchement récursif qui utilise une fonction de branchement définie sur un modèle vide. Le comportement de cet

opérateur, noté $\circlearrowleft_\emptyset$, est trivial ; aucun automate n'est rajouté ou modifié, et le module reste le même. L'opérateur $\circlearrowleft_\emptyset$ est donc l'identité sur les modules.

Enfin et par cohérence avec les définitions précédentes, nous définissons le module vide, noté $M_\emptyset$, comme le module défini sur les ensembles d'automates et d'entrées $S = I = \emptyset$.

5.3 Résultats

5.3.1 Universalité des branchements

Les opérateurs union et identité que nous décrivons dans la section précédente vérifient toutes le propriétés que l'on attend de tels opérateurs. Elles sont englobées par la propriété suivante.

Propriété 9. *Soient M, M' et M'' trois modules. Les équations suivantes sont vérifiées :*

$$\circlearrowleft_\emptyset M = M \tag{5.1}$$
$$M \cup M_\emptyset = M \tag{5.2}$$
$$M \cup M' = M' \cup M \tag{5.3}$$
$$(M \cup M') \cup M'' = M \cup (M' \cup M'') \tag{5.4}$$

Démonstration. Pour M un module, on considère le module $\circlearrowleft_\emptyset M$. Pour tout automate s, la fonction locale de s dans $\circlearrowleft_\emptyset M$ est définie par $f'_s(x \cdot i) = f_s((x \cdot i) \circ \hat{\mathtt{w}})$, où $\mathtt{w}$ est la fonction de branchement vide et f_s est la fonction locale de s dans M. Comme $\mathrm{dom}(\mathtt{w}) = \emptyset$, on observe $\hat{\mathtt{w}} = \mathrm{id}$ ce qui implique $f'_s = f_s$ et l'équation 5.1 est donc vérifiée.

Pour M un module, le module $M \cup M_\emptyset$ est défini sur l'ensemble d'automates $S \cup \emptyset = S$ et l'ensemble d'entrées $I \cup \emptyset = I$ tel que, pour tout automate s dans S, la fonction locale de s dans $M \cup M_\emptyset$ est définie comme $f'_s(x \cdot i) = f_s(x|_S \cdot i|_I) = f_s(x \cdot i)$, pour f_s la fonction locale de s dans M. On en déduit l'équation 5.2.

Soient M et M' deux modules. Le module $M \cup M'$ a pour ensemble d'automates $S \cup S'$ et pour ensemble d'entrées $I \cup I'$ tel que, pour tout automate $s \in S$, le module définit la fonction locale $g_s(x \cdot i) = f_s(x|_S \cdot i_I)$, et pour tout automate $s' \in S'$, le module définit la fonction locale $g_{s'}(x \cdot i) = f'_{s'}((x \cdot i|_{I'}) \circ \hat{\mathtt{w}})$. Cependant, comme la fonction de branchement considérée a pour domaine l'ensemble vide, on en déduit de nouveau que $\hat{\mathtt{w}} = \mathrm{id}$, ce qui implique que

$$\begin{aligned} g_{s'}(x \cdot i) &= f'_{s'}((x \cdot i|_{I'})|_{S' \cup I' \setminus \mathrm{dom}(\mathtt{w})}) \\ &= f'_{s'}((x \cdot i|_{I'})|_{S' \cup I'}) \\ &= f'_{s'}(x|_{S'} \cdot i|_{I'}). \end{aligned}$$

Maintenant, considérons le module $M' \cup M$. Ce module a pour ensemble d'automates $S' \cup S = S \cup S'$, et pour ensemble d'entrées $I' \cup I = I \cup I'$. Pour tout automate $s' \in S'$, le module définit la fonction locale $h_{s'}(x \cdot i) = f'_{s'}(x|_{S'} \cdot i|_{I'})$ et, pour tout automate $s \in S$, la fonction locale de s dans $M' \cup M$ vérifie

$$\begin{aligned} h_s(x \cdot i) &= f_s((x \cdot i|_I) \circ \hat{\mathtt{w}}) \\ &= f_s((x \cdot i|_I)|_{S \cup I \setminus \mathrm{dom}(\mathtt{w})}) \\ &= f_s((x \cdot i|_I)|_{S \cup I}) \\ &= f_s(x|_S \cdot i|_I). \end{aligned}$$

On en déduit l'équation 5.3.

En nous fondant sur les développements du paragraphe précédent, on observe que l'union de deux modules M et M' est un module qui est défini sur l'union des ensembles d'automates et des ensembles d'entrées de ces deux modules, et qui vérifie

$$f''_s(x \cdot i) = \begin{cases} f_s(x|_S \cdot i|_I) & \text{si } s \in S \\ f'_s(x|_{S'} \cdot i|_{I'}) & \text{si } s \in S' \end{cases}.$$

On en déduit que le module $(M \cup M') \cup M''$ est le module défini sur les ensembles $S \cup S' \cup S''$ et $I \cup I' \cup I''$ tel que

$$g_s(x \cdot i) = \begin{cases} f_s(x|_S \cdot i|_I) & \text{si } s \in S \\ f'_s(x|_{S'} \cdot i|_{I'}) & \text{si } s \in S' \\ f''_s(x|_{S''} \cdot i|_{I''}) & \text{si } s \in S'' \end{cases}.$$

Par le même développement et en interchangeant M et M'', le module $M \cup (M' \cup M'')$ est défini de la même façon, ce qui implique l'équation 5.4. □

On peut de plus observer que tout branchement non récursif peut être reproduit par la succession d'une union de modules et d'un branchement récursif, fait caractérisé par la propriété suivante.

Propriété 10. *Soient M, M' deux modules et $\mathtt{w}$ une fonction de branchement qui définit $M \rightarrowtail_{\mathtt{w}} M'$. L'équation suivante est vérifiée :*

$$M \rightarrowtail_{\mathtt{w}} M' = \circlearrowleft_{\mathtt{w}} (M \cup M').$$

Démonstration. Soient M et M' deux modules et $\mathtt{w} : I' \nrightarrow S$ une fonction de branchement. On observe que la fonction de branchement $\mathtt{w}$, définie comme fonction partielle de I' vers S, est également définie comme une fonction partielle de $I \cup I'$ vers $S \cup S'$. Ainsi elle est une fonction de branchement adéquate pour opérer le branchement $\circlearrowleft_{\mathtt{w}} (M \cup M')$.

Pour s dans S, la fonction locale définie par $M \rightarrowtail_{\mathtt{w}} M'$ est la fonction $g_s(x \cdot i) = f_s(x|_S \cdot i|_I)$. Pour s' dans S', cette fonction locale devient $g_{s'}(x \cdot i) = f'_{s'}((x \cdot i) \circ \hat{\mathtt{w}})$.

Dans le cas du module $\circlearrowleft_{\mathtt{w}}(M \cup M')$, la fonction locale de l'automate s dans S est $h_s(x \cdot i) = f''_s((x \cdot i) \circ \hat{\mathtt{w}})$, pour f''_s la fonction locale de s dans $M \cup M'$. Cela est égal à $f_s((x|_S \cdot i|_I) \circ \hat{\mathtt{w}})$, ce qui, par définition de $\mathtt{w}$, est égal à $f_s(x|_S \cdot i|_I)$. Pour tout automate s' dans S', la fonction locale de s' dans $\circlearrowleft_{\mathtt{w}}(M \cup M')$ est, par le même raisonnement, $h_{s'}(x \cdot i) = f'_{s'}((x|_{S'} \cdot i|_{I'}) \circ \hat{\mathtt{w}})$, ce qui est équivalent à $h_{s'}(x \cdot i) = f'_{s'}((x \cdot i) \circ \hat{\mathtt{w}})$, d'où le résultat. □

Les opérations d'union et de branchement récursif sont assez expressives pour permettre la construction de tout module à partir de modules de taille un. Cette expressivité est capturée par le théorème suivant.

Théorème 4 (Universalité des branchements)**.** *Soit M un module. Il existe un ensemble $\{M_1, M_2, \ldots, M_k\}$ de modules de taille 1 et $\mathtt{w}$ une fonction de branchement tels que*

$$M = \circlearrowleft_{\mathtt{w}} \bigcup_{i=0}^{k} M_k.$$

Démonstration. Pour prouver ce résultat, nous construisons l'ensemble de modules de taille 1 à partir des fonctions locales du module M.

Premièrement, pour tout s dans S, nous définissons la variable d'entrée α_s telle que $\alpha_s \notin I$. L'ensemble de ces variables constitue un ensemble I' distinct de l'ensemble I. De plus, nous définissons $\mu_s : \{s\} \cup I \cup I' \to S \cup I$ la fonction définie telle que

$$\mu(r) = \begin{cases} r & \text{si } r = s \\ r & \text{si } r \in I \\ q & \text{si } r \in I' \text{ et } r = \alpha_q \end{cases}.$$

Pour chaque automate s dans S de nouveau, nous définissons le module M_s comme le module défini sur l'ensemble d'automates $\{s\}$ et l'ensemble d'entrées $I \cup I'$ tel que la fonction locale de s est définie par $f^s_s(x \cdot i) = f_s((x \cdot i) \circ \mu_s)$.

Pour M un module, nous prenons l'ensemble de modules $\{M_s | s \in S\}$ et la fonction de branchement $\mathtt{w} : I \cup I' \twoheadrightarrow S$ définie sur le domaine $\mathrm{dom}(\mathtt{w}) = I'$ et telle que $\mathtt{w}(\alpha_s) = s$ pour tout automate s. Nous vérifions maintenant que la recomposition de ces éléments nous redonne le module de base.

Pour tout s dans S, la fonction locale de s dans $\bigcup\{M_s | s \in S\}$ est $f^s_s(x \cdot i) = f_s((x \cdot i) \circ \mu_s)$. Dans le module $\circlearrowleft_{\mathtt{w}} \bigcup\{M_s | s \in S\}$ qui est défini sur les ensembles S et I, cette fonction locale devient $f'_s(x \cdot i) = f^s_s((x \cdot i) \circ \hat{\mathtt{w}}) = f_s((x \cdot i) \circ \mu_s \circ \hat{\mathtt{w}})$.

On observe que, par définition, $\mu_s|_I = id$, et $\hat{\mathtt{w}}|_{S \cup I} = id$. De plus, $(\mu_s \circ \hat{\mathtt{w}})|_{I'} = id$. On en déduit que $\mu_s \circ \hat{\mathtt{w}} = id$, et donc que $f'_s(x \cdot i) = f_s(x \cdot i)$, ce qui implique $M = \bigcup\{M_s | s \in S\}$. □

5.3.2 Effets des branchements sur la dynamique

Les branchements sont des opérations simples qui modifient la description des modules sur lesquels ils opèrent. Ces modifications entraînent inévitablement des modifications sur la dynamique de ces modules. Dans les quelques résultats suivants, nous nous intéressons à observer les effets des différents types de branchement sur la dynamique des modules étudiés.

L'étude de l'évolution de la dynamique d'un module requiert une méthode de comparaison entre dynamiques. Ici, nous nous concentrons sur la dynamique totale des modules, et nous définissons une relation d'inclusion spécifique à ces graphes permettant d'étendre la définition d'inclusion sur les graphes. Afin de permettre cette définition, nous introduisons la notation $D(M)$ qui décrit le graphe de la dynamique totale du module M.

Définition 58 (Inclusion de dynamiques totales)**.** *Soient M et M' deux modules, tels que $S \subseteq S'$ et $I \subseteq I'$. On définit $D(M) \subseteq D(M')$ si et seulement si, pour toute arête $(x, (\delta, i), y)$ dans $D(M)$, il existe une arête $(x', (\delta', i'), y')$ dans $D(M')$ qui vérifie $x'|_S = x$, $y'|_S = y$, $\delta' \cap S = \delta$, et $i'|_I = i$.*

Nous définissons également l'extension intuitive du produit cartésien de deux graphes de la dynamique totale.

Définition 59 (Produit cartésien de dynamiques totales)**.** *Soient M et M' deux modules. Nous définissons le produit cartésien $D(M) \times D(M')$ comme le graphe étiqueté défini sur l'ensemble de sommets $\Lambda^{S \cup S'}$, et l'ensemble d'étiquettes $\wp(S \cup S') \times \Lambda^{I \cup I'}$, tel que le graphe $D(M) \times D(M')$ possède l'arête $(x \cdot x', (\delta \cup \delta', i \cdot i'), y \cdot y')$ si et seulement si le graphe $D(M)$ possède l'arête $(x, (\delta, i), y)$ et le graphe $D(M')$ possède l'arête $(x', (\delta', i'), y')$.*

Les effets des opérations d'union et de branchement récursif vide sur la dynamique des modules sont triviaux, et explicités par la propriété suivante.

Propriété 11 (Effet dynamique des branchements vides)**.** *Soient M, M' deux modules. Les équations suivantes sont vérifiées :*

$$D(\circlearrowleft_{\emptyset} M) = D(M) \tag{5.5}$$
$$D(M \cup M') = D(M) \times D(M'). \tag{5.6}$$

Démonstration. L'équation 5.5 se vérifie par l'application de la propriété 9.

Soient M et M' deux modules. Le module $M \cup M'$ est défini sur les automates $S \cup S'$ et les entrées $I \cup I'$ tel que, pour tout automate S, la fonction locale de s dans $M \cup M'$ vérifie

$$f''_s(x \cdot i) = \begin{cases} f_s(x|_S \cdot i|_I) & \text{si } s \in S \\ f'_s(x|_{S'} \cdot i|_{I'}) & \text{si } s \in S' \end{cases}$$

pour f_s et f'_s les fonctions locales de s dans M et M' respectivement.

Le graphe de la dynamique $D(M \cup M')$ est défini sur les sommets $\Lambda^{S \cup S'}$ et les étiquettes $\wp(S \cup S') \times \Lambda^{I \cup I'}$, et tel que $(x \cdot x', (\delta \cup \delta', i \cdot i'), y \cdot y')$ est une arête du graphe si et seulement si $(M \cup M')_{\delta \cup \delta'}(x \cdot x' \cdot i \cdot i') = y \cdot y'$. On procède aux étapes logiques suivantes :

$$
\begin{aligned}
& (M \cup M')_{\delta \cup \delta'}(x \cdot x' \cdot i \cdot i') = y \cdot y' \\
\iff & \forall s \in S \cup S', (M \cup M')_{\delta \cup \delta'}(x \cdot x' \cdot i \cdot i')_s = (y \cdot y')_s \\
\iff & (\forall s \in S, (M \cup M')_{\delta \cup \delta'}(x \cdot x' \cdot i \cdot i')_s = y_s) \\
& \wedge (\forall s' \in S', (M \cup M')_{\delta \cup \delta'}(x \cdot x' \cdot i \cdot i')_{s'} = y'_{s'}) \\
\iff & \forall s \in S, \begin{cases} f''_s(x \cdot x' \cdot i \cdot i') & \text{si } s \in \delta \cup \delta' \\ (x \cdot x')_s & \text{sinon} \end{cases} = y_s \\
& \wedge \forall s' \in S', \begin{cases} f''_{s'}(x \cdot x' \cdot i \cdot i') & \text{si } s' \in \delta \cup \delta' \\ (x \cdot x')_{s'} & \text{sinon} \end{cases} = y'_{s'} \\
\iff & \forall s \in S, \begin{cases} f_s((x \cdot x')|_S \cdot (i \cdot i')|_I) & \text{si } s \in \delta \\ x_s & \text{sinon} \end{cases} = y_s \\
& \wedge \forall s' \in S', \begin{cases} f'_{s'}((x \cdot x')|_{S'} \cdot (i \cdot i')|_{I'}) & \text{si } s' \in \delta' \\ x'_{s'} & \text{sinon} \end{cases} = y'_{s'} \\
\iff & \forall s \in S, \begin{cases} f_s(x \cdot i) & \text{si } s \in \delta \\ x_s & \text{sinon} \end{cases} = y_s \\
& \wedge \forall s' \in S', \begin{cases} f'_{s'}(x' \cdot i') & \text{si } s' \in \delta' \\ x'_{s'} & \text{sinon} \end{cases} = y'_{s'} \\
\iff & (\forall s \in S, M_\delta(x \cdot i)_s = y_s) \wedge (\forall s' \in S', M'_{\delta'}(x' \cdot i')_{s'} = y'_{s'}) \\
\iff & (M_\delta(x \cdot i) = y) \wedge (M'_{\delta'}(x' \cdot i') = y').
\end{aligned}
$$

Il résulte de ce développement qu'il existe une arête $(x \cdot x', (\delta \cup \delta', i \cdot i'), y \cdot y')$ dans la dynamique $D(M \cup M')$ si et seulement si le graphe $D(M)$ accepte l'arête $(x, (\delta, i), y)$ et si le graphe $D(M')$ accepte l'arête $(x', (\delta', i'), y')$. Ainsi, le graphe $D(M \cup M')$ coïncide avec la définition du graphe $D(M) \times D(M')$, et l'équation 5.6 est vérifiée. □

Lorsqu'un réseau est altéré par un branchement récursif, le module résultant est identique au réseau initial à la différence près que l'influence de certaines entrées est maintenant remplacée par l'influence d'automates. Cela signifie en particulier que tous les comportements du réseau final peuvent être reproduits dans le réseau initial en prenant les séquences d'entrées qui miment le comportement des automates concernés par le branchement. Cette observation implique que le branchement récursif ne peut que réduire la dynamique des réseaux étudiés ; plus précisément, la dynamique du nouveau réseau est strictement incluse dans la dynamique initiale.

Propriété 12 (Effet dynamique du branchement récursif)**.** *Un module M et une fonction de branchement non vide* `w` *vérifient*

$$D(\circlearrowleft_{\mathtt{w}} M) \subsetneq D(M).$$

Démonstration. Soit M un module et $\mathtt{w}$ une fonction de branchement. Les graphes $D(M)$ et $D(\circlearrowleft_{\mathtt{w}} M)$ sont tous deux définis sur le même ensemble de sommets Λ^S. Le graphe $D(M)$ accepte ses étiquettes dans l'ensemble $\wp(S) \times \Lambda^I$, là où le graphe $D(\circlearrowleft_{\mathtt{w}} M)$ accepte ses étiquettes dans l'ensemble $\wp(S) \times \Lambda^{I \setminus \mathrm{dom}(\mathtt{w})}$.

Soit $(x, (\delta, i), y)$ une arête dans $D(\circlearrowleft_{\mathtt{w}} M)$. On définit i' la configuration d'entrée définie dans Λ^I telle que

$$i'_\alpha = \begin{cases} x_{\mathtt{w}(\alpha)} & \text{si } \alpha \in \mathrm{dom}(\mathtt{w}) \\ i_\alpha & \text{sinon} \end{cases}.$$

On observe que, par définition du branchement récurif $\mathtt{w}$, l'arête $(x, (\delta, i'), y)$ est une arête du graphe $D(M)$. Comme $i'|_{I \setminus \mathrm{dom}(\mathtt{w})} = i$, $D(\circlearrowleft_{\mathtt{w}} M) \subseteq D(M)$. Enfin, comme le branchement $\mathtt{w}$ n'est pas vide, $D(\circlearrowleft_{\mathtt{w}} M) \neq D(M)$, ce qui implique le résultat. □

Cette propriété est illustrée en figure 5.10 sur la dynamique totale du réseau d'automate obtenu dans l'exemple 41. La dynamique illustrée en figure 5.10 est incluse dans la dynamique illustrée dans les figures 5.5 et 5.6.

Lorsque l'on combine deux modules par le biais d'un branchement non récursif, la dynamique obtenue est une sous-dynamique du produit cartésien des dynamiques des réseaux initiaux.

Propriété 13 (Effet dynamique du branchement non récursif)**.** *Deux modules M, M' et une fonction de branchement non vide $\mathtt{w}$ vérifient*

$$D(M \rightarrowtail_{\mathtt{w}} M') \subsetneq D(M) \times D(M').$$

Démonstration.

$$\begin{aligned} D(M \rightarrowtail_{\mathtt{w}} M') &= D(\circlearrowleft_{\mathtt{w}} (M \cup M')) && \text{(Propriété 10)} \\ &\subsetneq D(M \cup M') && \text{(Propriété 12)} \\ &= D(M) \times D(M'). && \text{(Propriété 11)} \end{aligned}$$

□

5.3.3 Modularité et simulation

Les modules sont une généralisation des réseaux d'automates booléens, qui sont eux-mêmes une généralisation des automates cellulaires. En cela, les modules possèdent une place forte pour proposer des résultats généraux de simulation sur les modèles de calcul naturel. Ainsi, nous proposons d'utiliser la modularité de ce modèle afin de construire un méta-théorème qui dérive des résultats très variés de simulation à partir d'une preuve de simulation locale.

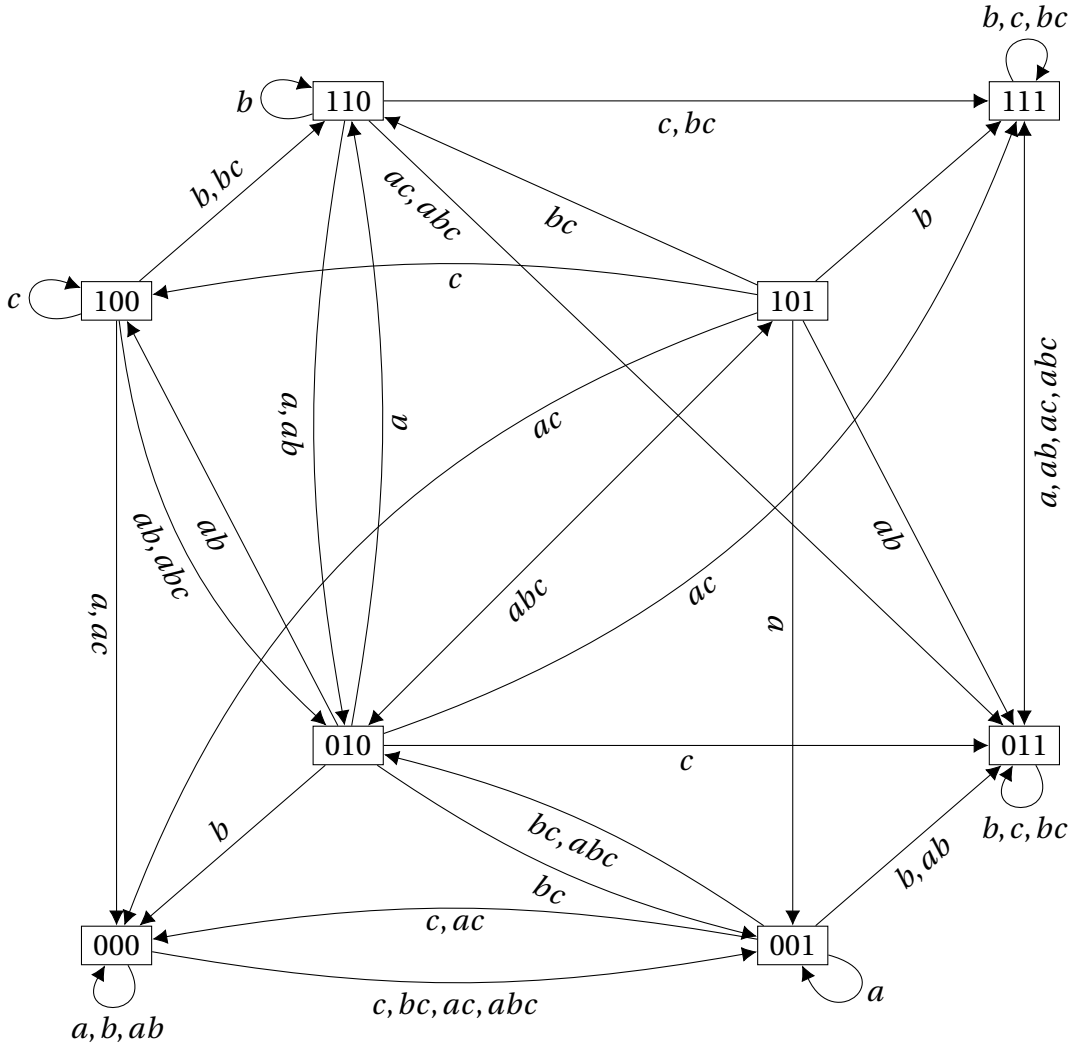

FIGURE 5.10 – Graphe de la dynamique totale du réseau d'automates booléen obtenu par le branchement récursif développé dans l'exemple 41. L'étiquette ab représente la mise à jour $\delta = \{a, b\}$.

La simulation locale nous permet de définir une famille de modules qui reproduisent le comportement local d'automates ou de cellules dans d'autres modèles. Ainsi, la composition de ces modules en un super-réseau de modules simule naturellement les réseaux dont les éléments sont localement simulés.

Plus formellement, supposons un réseau d'automates F composé d'un ensemble d'automates S à états dans l'ensemble Λ. Nous simulons localement chaque automate a dans S par un module M_a, défini sur un ensemble d'automates S_a et un ensemble d'entrées I_a. Chaque module M_s utilise un ensemble d'états Λ'.

Le module M_a représente l'automate a dans notre composition finale. Ainsi, la configuration de M_a a pour mission d'encoder l'état de a à tout moment. Cela est

exprimé par le biais de ce que nous appellons un encodage.

Définition 60 (Encodage)**.** *Soit R un ensemble d'automates, et Λ, Λ' deux ensembles d'états. Un* encodage *sur R de Λ' vers Λ est une fonction partielle surjective $\phi : \Lambda'^R \twoheadrightarrow \Lambda \cup \{\bullet\}$.*

Exemple 43. *Soit les ensembles $Q = \{a, b\}$, $\Lambda' = \{0, 1, 2\}$ et $\Lambda = \{0, 1\}$. On définit ϕ l'encodage sur Q de Λ' vers Λ défini tel que*

$$\phi(00) = 0$$
$$\phi(01) = 1$$
$$\phi(10) = 2$$
$$\phi(11) = \bullet$$

La valeur $\bullet$ signifie que la configuration considérée n'a pas de transcription dans l'encodage considéré.

Dans notre construction, chaque module M_a joue le rôle d'un automate de M. Nous nommons ϕ_a l'encodage sur S_a de Λ' vers Λ, pour chaque a dans S.

Si, dans M, les différents automates interagissent par influence, alors les modules $\{M_s | s \in S\}$ doivent être interconnectés. Pour toute paire d'automates distincts a, b dans S, nous définissons le branchement $\mathtt{w}_{a,b} : I_b \nrightarrow S_a$ qui précise la façon dont le module M_a est branché sur le module M_b. Ici, l'ensemble $\mathrm{codom}(\mathtt{w}_{a,b})$ sert d'interface de communication de a vers b. Avant que le branchement soit opéré, cette interface est représentée virtuellement par l'ensemble des entrées décrit par $\mathrm{dom}(\mathtt{w}_{a,b})$. Les états de ces nœuds et de ces entrées permettent au module M_a de transférer son état à l'automate M_b. Afin de s'assurer que cette communication soit cohérente avec les encodages précédemment définis, nous introduisons l'encodage $\phi_{a,b}$ défini sur $\mathrm{codom}(\mathtt{w}_{a,b})$ de Λ' vers Λ, pour chaque paire d'automates distincts a, b dans S. Dans la pratique, la valeur donnée par l'encodage $\phi_{a,b}$ devra être cohérente avec la valeur de l'encodage ϕ_a, pour tout b différent de a dans S.

En condition supplémentaire, il est nécessaire que les fonctions de branchement soient injectives. Ainsi, nous définissons la fonction $\mathtt{w}^{-1}_{a,b} : \mathrm{codom}(\mathtt{w}_{a,b}) \to \mathrm{dom}(\mathtt{w}_{a,b})$ telle que $\mathtt{w}^{-1}_{a,b} \circ \mathtt{w}_{a,b} = id$. Nous pouvons maintenant définir la simulation locale d'une fonction de notre réseau d'automates F par un module.

Définition 61 (Simulation locale de réseau d'automates)**.** *Pour F un réseau d'automates, a un automate dans S et Δ un mode de mise à jour de M_a, le module M_a* simule localement *f_a, noté $f_a \preccurlyeq_\Delta M_a$, si et seulement si pour toute configuration $x \in \Lambda^S$,*

- *et pour toute configuration $x' \in \Lambda'^{S_a}$ telle que $\phi_a(x') = x_a$,*
- *et pour toute configuration d'entrée $i' : \Lambda'^{I_a}$ telle que pour tout b dans S, si $a \neq b$ alors $\phi_{b,a}(i' \circ \mathtt{w}^{-1}_{b,a}) = x_b$*

on a

$$\phi_a(M_{a\Delta}(x' \cdot i')) = f_a(x).$$

Lorsque les modules $\{M_s | s \in S\}$ sont branchés en un seul réseau, il est primordial que les modules de mise à jour Δ considérés ne parasitent pas le fonctionnement du réseau global. Pour cela, nous posons la condition que tous mode de mise à jour utilisé sur un sous-réseau M_s doit mettre à jour au moins tout les nœuds qui possèdent des entrées à la première étape de mise à jour. Lorsque cela est fait, il n'est pas possible qu'un module change son encodage avant qu'un autre module ait pu acquérir la valeur de son encodage à temps.

Définition 62 (Mode de mise à jour entrée-prioritaire)**.** *Soit Δ un mode de mise à jour pour un module M. Le mode de mise à jour Δ est dit entrée-prioritaire si et seulement si, pour tout $k > 1$, $s \in \Delta_k$ implique que s n'est influencé par aucune entrée.*

Nous pouvons maintenant introduire notre définition spécialisée de simulation entre réseaux d'automates.

Définition 63 (Simulation entre réseaux d'automates)**.** *Soient F, F' deux réseaux d'automates, définis sur des ensembles d'états Λ et Λ'. Le réseau F' simule le réseau F, noté $F \preccurlyeq F'$, si et seulement s'il existe une fonction $\Phi : \Lambda'^{S'} \to \Lambda^S \cup \{\bullet\}$ dite encodage global telle que, pour toute mise à jour $\delta \subset S$, il existe un mode de mise à jour fini Δ' tel que pour toutes configurations x de F et x' de F' telles que $\Phi(x') = x$,*

$$\Phi(F'_{\Delta'}(x')) = F_\delta(x).$$

Depuis un réseau F, et en définissant l'ensemble de modules $\{M_s | s \in S\}$, de branchements $\{\mathtt{w}_{a,b} | a, b \in S \,|a \neq b\}$ ainsi que les encodages $\{\phi_s | s \in S\}$ et $\{\phi_{a,b} | a, b \in S | a \neq b\}$, nous proposons le théorème suivant.

Théorème 5. *Soit $\mathtt{w}$ le branchement défini comme la concaténation des branchements $\{\mathtt{w}_{a,b} | a, b \in S \,|a \neq b\}$. Si, pour tout $s \in S$, le module M_s simule localement la fonction f_s, et si $\mathtt{w}$ est une fonction de branchement proprement définie qui associe un sommet à tous les sommets dans l'union des modules $\{M_s | s \in S\}$, alors*

$$F \preccurlyeq \circlearrowleft_{\mathtt{w}} \left(\bigcup_{s \in S} M_s \right).$$

Démonstration. Nous allons d'abord démontrer que le comportement du réseau F' peut être décrit comme la somme des comportements des modules $\{M_s | s \in S\}$ pris séparément. Puis nous utiliserons la propriété de simulation locale de chaque module pour extrapoler le comportement global de simulation du réseau F'.

Soit $F' = \circlearrowleft_{\mathtt{w}} (\bigcup_{s \in S} M_s)$. Par l'énoncé de notre théorème, F' est un réseau d'automates. Soit T son ensemble d'automates, défini comme l'union des ensembles d'automates des modules $\{M_s | s \in S\}$. Pour tout a dans S, tout mode de mise à jour entrée-prioritaire Δ pour le module M_a, pour tout mode de mise à jour Δ' sur $T \setminus M_a$, et pour toute configuration x, nous énonçons que

$$F'_{\Delta \cup \Delta'}(x)|_{S_a} = M_{a\Delta}(x|_{S_a} \cdot (x \circ \mathtt{w}|_{I_a})). \tag{5.7}$$

Pour voir que cette équation est vraie, considérons qu'à la première étape de l'exécution, le branchement $\mathtt{w}$ implique que pour tout s dans S_a, et toute configuration x, $F'(x)_s = (\bigcup_{s\in S} M_s)(x\cdot(x\circ\mathtt{w}))$. En particulier, cela signifie que $F'(x)_s = M_a(x|_{S_a}\cdot(x\circ\mathtt{w}|_{I_a}))$.

Considérons A le sous-ensemble des automates de S_a qui sont influencés par au moins une entrée, et $B = S_a \setminus A$ l'ensemble des automates qui ne sont influencés par aucune entrée. Par la nature entrée-prioritaire de Δ, nous savons que si $s \in \Delta_k$ et $k > 1$, alors $s \in B$.

Considérons d'abord la mise à jour des automates dans A. Nous posons $\delta = \Delta_1$ et $\delta' = \Delta'_1$. Nous pouvons clairement déduire de la proposition précédente que

$$F'_{\delta\cup\delta'}(x)|_A = M_{a\delta}(x|_{S_a}\cdot(x\circ\mathtt{w}|_{I_a}))|_A.$$

De plus, il n'existe aucun s dans A tel que s est également dans Δ_k pour $k > 1$. Nous pouvons donc conclure, puisqu'aucune fonction locale de A n'est mise à jour dans le reste de l'exécution, que $F'_{\delta\cup\delta'}(x)|_A = F'_{\Delta\cup\Delta'}(x)|_A$, et que $M_{a\delta}(x|_{S_a}\cdot(x\circ\mathtt{w}|_{I_a}))|_A = M_{a\Delta}(x|_{S_a}\cdot(x\circ\mathtt{w}|_{I_a}))|_A$. En conclusion de notre travail sur l'ensemble A,

$$F'_{\Delta\cup\Delta'}(x)|_A = M_{a\Delta}(x|_{S_a}\cdot(x\circ\mathtt{w}|_{I_a}))|_A.$$

Nous considérons maintenant la mise à jour de l'ensemble d'automates B. Pour tout $s \in B$, nous savons que $F'(x)_s = M_a(x|_{S_a}\cdot(x\circ\mathtt{w}|_{I_a}))_s$. Par la définition de B, l'automate s n'est influencé par aucune entrée. Cela signifie que, pour toutes mises à jour $\delta \subseteq S_a$ et $\delta' \subseteq T\setminus S_a$ et pour toute configuration d'entrée $i \in \Lambda'^{I_a}$, on a $F'_{\delta\cup\delta'}(x)|_B = M_{a\delta}(x|_{S_a}\cdot i)|_B$. Par un simple raisonnement récursif cela implique que $F'_{\Delta\cup\Delta'}(x)|_B = M_{a\Delta}(x|_{S_a}\cdot i)|_B$.

En rassemblant nos raisonnements concernant les ensembles A et B, nous obtenons que $F'_{\Delta\cup\Delta'}(x) = M_{a\Delta}(x|_{S_a}\cdot(x\circ\mathtt{w}|_{I_a}))|_A\cdot M_{a\Delta}(x|_{S_a}\cdot i)|_B$, pour toute configuration d'entrée i. Cela signifie qu'en prenant en particulier $i = x\circ\mathtt{w}|_{I_a}$, on vérifie la proposition 5.7.

Nous considérons à présent la fonction $\Phi : \Lambda'^T \to \Lambda^S \cup \{\bullet\}$ qui vérifie que, pour tout $x' \in \Lambda'^T$, $\Phi(x') = \bullet$ s'il existe $a \in S$ tel que $\phi_a(x|_{S_a}) = \bullet$, et $\Phi(x')_a = \phi_a(x|_{S_a})$ dans le cas contraire. Soient x et x' tels que $\Phi(x') = x$ et $x \neq \bullet$. Soit $\delta \subseteq S$ une mise à jour de F. Pour tout a dans δ, nous définissons Δ'_a un mode de mise à jour entrée-prioritaire avec lequel M_a simule localement la fonction f_a, ce qui est toujours possible par hypothèse du théorème. Nous définissons le mode de mise à jour Δ' sur T tel que $\Delta' = \bigcup\{\Delta'_a | a \in \delta\}$.

Nous allons maintenant prouver que $\Phi(F'_{\Delta'}(x')) = F_\delta(x)$. Premièrement, nous observons que

$$F'_{\Delta'}(x') = \mathrm{concat}(F'_{\Delta'}(x')|_{S_a} | a \in S)$$

ce qui se développe en

$$F'_{\Delta'}(x') = \mathrm{concat}(F'_{\Delta'}(x')|_{S_a} | a \in \delta)\cdot\mathrm{concat}(x'|_{S_a} | a \in S\setminus\delta)$$

ce dont nous déduisons que

$$\begin{aligned} & F'_{\Delta'}(x') = \operatorname{concat}(F'_{\Delta'_a \cup \bigcup_{b\in\delta, b\neq a}\Delta'_b}(x')|_{S_a} | a \in \delta) \cdot \operatorname{concat}(x'|_{S_a} | a \in S\setminus\delta) \\ \Longrightarrow\ & F'_{\Delta'}(x') = \operatorname*{concat}_{a\in\delta}(M_{a\Delta'_a}(x'|_{S_a} \cdot (x \circ \mathtt{w}|_{I_a}))) \cdot \operatorname*{concat}_{a\in S\setminus\delta}(x'|_{S_a}) \qquad \text{(Équation 5.7)} \end{aligned}$$

Comme le résultat de l'exécution du module M_a est une configuration sur S_a, nous pouvons déduire la valeur de $\Phi(F'_{\Delta'}(x))$ suivante :

$$\Phi(F'_{\Delta'}(x'))_a = \begin{cases} \phi_a(M_{a\Delta_a}(x'|_{S_a} \cdot (x \circ \mathtt{w}|_{I_a}))) & \text{si } a \in \delta \\ \phi_a(x'|_{S_a}) & \text{si } a \in S\setminus\delta \end{cases}.$$

Par définition de x et x', nous savons que $\phi_a(x'|_{S_a}) = x_a$ et que $\phi_{b,a}(x' \circ \mathtt{w}_{b,a} \circ \mathtt{w}_{b,a}^{-1}) = \phi_{b,a}(x'|_{\operatorname{codom}(\mathtt{w}_{b,a})}) = \phi_b(x'|_{S_b}) = x_b$, par la définition de $\phi_{b,a}$. De cela, nous appliquons la définition de la simulation locale et obtenons

$$\Phi(F'_{\Delta'}(x'))_a = \begin{cases} f_a(\Phi(x')) & \text{si } a \in \delta \\ \phi_a(x'|_{S_a}) & \text{si } a \in S\setminus\delta \end{cases} = \begin{cases} f_a(\Phi(x)) & \text{si } a \in \delta \\ \Phi(x')_a & \text{si } a \in S\setminus\delta \end{cases}.$$

De plus, par la définition de la mise à jour de F, on obtient que

$$F_\delta(x)_a = \begin{cases} f_a(x) & \text{si } a \in \delta \\ x_a & \text{si } a \in S\setminus\delta \end{cases}.$$

Enfin, par l'hypothèse que $x = \Phi(x')$, nous obtenons

$$F_\delta(x)_a = \begin{cases} f_a(\Phi(x')) & \text{si } a \in \delta \\ \Phi(x')_a & \text{si } a \in S\setminus\delta \end{cases},$$

ce qui implique le résultat.

□

Ce théorème permet la généralisation de résultats simples de simulation, et est présenté ici principalement pour sa qualité d'illustration du formalisme des modules en tant que modèle de décomposition et recomposition des réseaux d'automates. Dans cet esprit, nous présentons trois résultats simples dérivés de l'application de notre théorème. Il s'agit de l'universalité intrinsèque des réseaux d'automates booléens, lire la capacité des réseaux d'automates booléens de simuler l'ensemble des réseaux d'automates, ainsi que la simulation de tout réseau d'automate booléen par un réseau d'automates booléen disjonctif, ainsi que par un réseau d'automates booléen localement monotone.

Corollaire 1. *Soit F un réseau d'automates défini sur l'ensemble d'états Λ. Il existe un réseau d'automates booléens F' avec au plus* $\log(|\Lambda|)$ *fois plus d'automates tel que $F \preccurlyeq F'$.*

Démonstration. Soit F un réseau défini sur l'ensemble d'états Λ. Dans cette preuve, nous construisons pour chaque automate a dans F un module booléen M_a qui encode l'état de a dans Λ à l'aide de $\log(|\Lambda|)$ automates avec état dans $\mathbb{B}$. Chacun de ces automates calcule la fonction locale de f_a et dérive le bit d'information qui l'intéresse, de façon à ce que l'ensemble des automates de M_a encodent dans leur ensemble l'état de a dans F.

Le module M_a est défini sur l'ensemble d'automates $S_a = \{a_0, a_1, \dots, a_{\log(|\Lambda|)-1}\}$. Pour tout automate b de F différent de a, on définit l'ensemble d'entrées $I_{b,a} = \{\alpha_{b_0,a}, \alpha_{b_1,a}, \dots, \alpha_{b_{\log(|\Lambda|)-1},a}\}$. L'ensemble des entrées de M_a est l'ensemble $I_a = \bigcup_b I_{b,a}$.

Soit $\mathrm{bit}_k(\lambda)$ la fonction qui, pour tout état λ, retourne la valeur du bit numéro k dans le nombre booléen de taille $\log(|\Lambda|)$ qui encode la valeur de λ. Cette fonction est canonique en supposant un ordre arbitraire sur les éléments de Λ, qui est un ensemble fini.

Pour a_k un automate de M_a, x une configuration sur S_a, on définit ϕ_a l'encodage qui à x associe la valeur dans Λ qui est cohérente avec la fonction bit, et renvoie $\bullet$ si aucune valeur n'est cohérente. En particulier, pour $\lambda \in \Lambda$, si $x_{a_k} = \mathrm{bit}_k(\lambda)$, alors $\phi_a(x) = \lambda$.

On définit le branchement $\mathtt{w}_{b,a}$ comme le branchement avec domaine $I_{b,a}$ et co-domaine S_b tel que $\mathtt{w}_{b,a}(\alpha_{b_k,a}) = b_k$. Le branchement $\mathtt{w}_{b,a}$ est injectif.

Pour x une configuration sur S_b, on définit l'encodage $\phi_{b,a}$ comme la fonction qui vérifie $\phi_{b,a}(x) = \lambda$ si pour tout k, $x_{b_k} = bit_k(\lambda)$, et $\phi_{b,a}(x) = \bullet$ si aucun λ n'existe.

Soient x et i des configurations d'automates et d'entrées du module M_a. Depuis ces éléments, on peut reconstruire une configuration y sur le réseau F. Pour ce faire, supposons qu'il existe un λ_a tel que $\phi_a(x) = \lambda_a$ et que, pour tout $b \neq a$, il existe un λ_b tel que $\phi_{b,a}(i) = \lambda_b$. La recomposition de cette configuration, que nous notons $\mathrm{recomp}(x,i)$ vérifie $\mathrm{recomp}(x,i)_s = \lambda_s$ dans ce cas. Dans le cas contraire, on évalue $\mathrm{recomp}(x,i)_s = \lambda_\bullet$, où $\lambda_\bullet$ est une valeur arbitraire dans Λ.

Enfin, la fonction locale f_{a_k} est définie par

$$f_{a_k}(x \circ i) = \mathrm{bit}_k(f_a(\mathrm{recomp}(x,i)).$$

Pour simuler localement la fonction f_a, le module M_a opère avec la mise à jour parallèle.

Par la structure de notre construction, il est trivial de vérifier que le module M_a simule localement la fonction f_a. Nous déduisons le résultat par application du Théorème 5. □

Corollaire 2. *Soit F un réseau d'automates booléens. Il existe un réseau d'automates booléens F' tel que chaque fonction locale de F' est une clause disjonctive, et $F \preccurlyeq F'$.*

Démonstration. Soit F un réseau d'automates booléens. Dans cette preuve, nous construisons, pour chaque automate a de F, un module M_a exclusivement composé de fonctions booléennes qui peuvent être écrites comme une clause disjonctive, qui simule f_a localement.

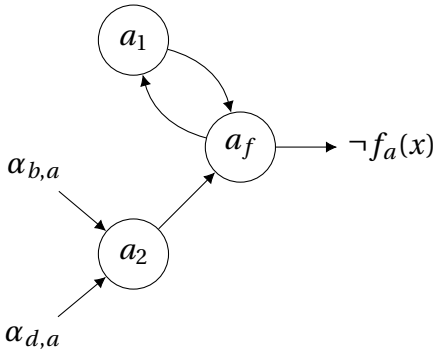

FIGURE 5.11 – Graphe d'interaction du module localement disjonctif qui simule localement la fonction $f_a(x) = x_a \wedge (\neg x_b \vee x_d)$. Nous nommons $C_{a,1} = x_a$ et $C_{a,2} = \neg x_b \vee x_d$ les clauses de la fonction f_a. Nous observons $f_{a_1} = \neg x_{a_f}$, $f_{a_2} = \neg \alpha_{b,a} \vee \alpha_{d,a}$ et $f_{a_f} = \neg x_{a_1} \vee \neg x_{a_2}$.

Premièrement, nous faisons l'observation que toute fonction booléenne peut être représentée par une formule booléenne sous la forme normale disjonctive. Cela signifie que la fonction f_a peut être réalisée par une formule $\mathbf{g}_a = \bigvee_{k=1}^{n_a} C_{a,k}$, pour n_a le nombre de clauses de la formule $\mathbf{g}_a$, et où chaque $C_{a,k}$ est une clause conjonctive.

Notre construction repose sur la composition d'un module M_a qui possède un nœud dénoté a_k pour chaque clause $C_{a,k}$, ainsi qu'un nœud dénoté a_f. Le module définit également un ensemble d'entrées $\alpha_{b,a}$ pour chaque automate b dans F différent de a. L'encodage ϕ_a est défini tel que $\phi_a(x) = x_{a_f}$, le branchement $\mathtt{w}_{b,a}$ a pour domaine $\{\alpha_{b,a}\}$ et co-domaine $\{x_{b_f}\}$ ce qui implique $\mathtt{w}_{b,a}(\alpha_{b,a}) = x_{b_f}$, et l'encodage $\phi_{b,a}(x|_{\{b_f\}}) = x_{b_f}$.

La fonction locale de a_k calcule la négation de la valeur de $C_{a,k}$. Pour ce faire, le littéral associé à a dans $\mathbf{g}_a$ est substitué par x_{a_f} et, pour tout automate b différent de a, le littéral associé à b dans $\mathbf{g}_a$ est substitué par $i_{\alpha_{b,a}}$. Comme $C_{a,k}$ est une clause conjonctive, sa négation est une clause disjonctive.

La fonction locale de a_f calcule la valeur de $\mathbf{g}_a$ par le biais du calcul $\bigvee_{k=1}^{n_a} C_{a,k}$, qui est opéré en subsituant la valeur de $C_{a,k}$ par la négation $\neg x_{a_k}$.

Pour simuler localement l'automate a, le module M_a opère sur le mode de mise à jour $\Delta_a = (\{a_1, a_2, \ldots, a_{n_a}\}, \{a_f\})$, qui est un mode de mise à jour entrée-prioritaire. Par notre construction, il est facile de vérifier que le module M_a simule localement la fonction f_a. Nous déduisons le résultat par application du Théorème 5. □

Nous illustrons un exemple de module constitué de fonctions disjonctives qui simule localement la fonction $f_a(x) = x_a \wedge (\neg x_b \vee x_d)$ en figure 5.11.

Nous définissons qu'une fonction locale f d'un réseau d'automates booléens est localement monotone si, pour toute paire de configurations x, y telles que $x \leq y$, $f(x) \leq f(y)$. Ici, l'opération de comparaison utilisée est la comparaison binaire bit à bit : la configuration 00 est inférieure à 10, 01 ou 11, mais 10 et 01 ne sont pas

comparables.

Corollaire 3. *Soit F un réseau d'automates booléens. Il existe un réseau d'automates booléens F' tel que chaque fonction locale de F' est une fonction localement monotone, et $F \preccurlyeq F'$.*

Démonstration. Soit F un réseau d'automates booléens. Dans cette preuve, nous construisons, pour chaque automate a de F, un module M_a, exclusivement composé de fonctions localement monotones, qui simule f_a.

Pour toute paire d'entrées x, y, une fonction localement monotone f vérifie $x \leq y \implies f(x) \leq f(y)$. Dans le cas des fonctions locales d'un réseau d'automates, nous définissons qu'une configuration x est inférieure ou égale à une configuration y définie sur les mêmes ensembles si et seulement si $x_s \leq y_s$ pour tout automate s.

Soit a un automate de F. Nous définissons le module M_a composé de deux automates a_0 et a_1, et un ensemble d'entrées I_a qui définit deux entrées $\alpha_{b,a,0}, \alpha_{b,a,1}$ pour chaque automate b différent de a. L'encodage ϕ_a vérifie $\phi_a(x) = 1$ si et seulement si $x_{a_0} = 0$ et $x_{a_1} = 1$, $\phi_a(x) = 0$ si et seulement si $x_{a_0} = 1$ et $x_{a_1} = 0$, et $\phi_a(x) = \bullet$ dans tous les autres cas. Le branchement injectif $\mathtt{w}_{b,a}$ a pour domaine $\{\alpha_{b,a,0}, \alpha_{b,a,1}\}$ et codomaine $\{b_0, b_1\}$ tel que $\mathtt{w}_{b,a}(\alpha_{b,a,k}) = b_k$ pour $k \in \mathbb{B}$. Enfin, l'encodage $\phi_{b,a}(x) = \phi_b(x)$.

Pour tous x et i tels que les encodages sont proprement définis, soit $\mathrm{recomp}(x, i)$ la configuration de F telle que $\mathrm{recomp}(x, i)_a = \phi_a(x)$ et $\mathrm{recomp}(x, i)_b = \phi_b(i \circ \mathtt{w}_{b,a}^{-1})$, pour tout $b \neq a$. Dans le cas où $\mathrm{recomp}(x, i)$ est défini, les fonctions locales de a_0 et a_1 vérifient $f_{a_1}(x \circ i) = f_a(\mathrm{recomp}(x, i))$ et $f_{a_0}(x \circ i) = \neg f_{a_1}(x \circ i)$.

Cet encodage n'est pas défini si et seulement s'il existe au moins une paire (x_{a_0}, x_{a_1}) ou $(\alpha_{b,a,0}, \alpha_{b,a,1})$ dont la valeur des éléments est identique. S'il existe une telle paire de valeur 0, on dit que (x, i) encode une paire 0. S'il existe une telle paire de valeur 1, on dit que (x, i) encode une paire 1. Dans le cas où (x, i) encode une paire 0 ou une paire 1, nous définissons l'évaluation des fonctions locales f_{a_0} et f_{a_1} comme

$$f_{a_0}(x \circ i) = f_{a_1}(x \circ i) = \begin{cases} 0 & \text{si } (x, i) \text{ n'encode pas de paire 1} \\ 1 & \text{si } (x, i) \text{ n'encode pas de paire 0} \\ 0 & \text{si } (x, i) \text{ encode au moins une paire 0 et une paire 1} \end{cases}.$$

Soient x et y deux configurations du module $\bigcup_{s \in S} M_s$. Si $x = y$ alors $f_{a_0}(x) = f_{a_0}(y)$ et $f_{a_1}(x) = f_{a_1}(y)$. Supposons que $x \leq y$ et $x \neq y$. Nous construisons (x', i') et (y', j') les paires de configurations d'automates et d'entrées équivalentes à x et y.

Supposons que (x', i') encode une paire 0, et pas de paire 1. Alors $f_{a_0}(x) = f_{a_1}(x) = 0$ ce qui est toujours inférieur ou égal à $f_{a_0}(y)$ et $f_{a_1}(y)$.

Supposons que (x', i') encode une paire 1 et pas de paire 0. Alors (y', j') encode au moins les même paires 1 et ne peut pas encoder de paire 0, ce qui implique que $f_{a_0}(y) = f_{a_1}(y) = 1$, ce qui est toujours supérieur ou égal à $f_{a_0}(x)$ et $f_{a_1}(x)$.

Supposons que (x', i') n'encode aucune paire 0 ni aucune paire 1. Cela signifie que (y', j') n'encode pas de paire 0 et encode au moins une paire 1, ce qui signifique que $f_{a_0}(y) = f_{a_1}(y) = 1$, ce qui est toujours supérieur ou égal à $f_{a_0}(x)$ et $f_{a_1}(x)$.

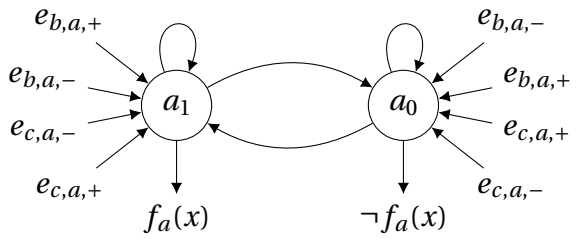

FIGURE 5.12 – Graphe d'interaction du module composé de fonctions localement monotones qui simule localement la fonction $f_a(x) = x_a \wedge (\neg x_b \vee x_c)$.

Supposons enfin que (x', i') encode au moins une paire 0 et au moins une paire 1. Cela signifie que $f_{a_0}(x) = f_{a_1}(x) = 0$, ce qui est toujours inférieur ou égal à $f_{a_0}(y)$ et $f_{a_1}(y)$.

En conclusion, les fonctions locales du module M_a sont localement monotones. Notre construction permet de vérifier que le module M_a simule localement la fonction f_a grâce au mode de mise à jour parallèle. Nous déduisons le résultat par application du théorème 5. □

Nous illustrons un exemple de module constitué de fonctions localement monotones qui simule localement la fonction $f_a(x) = x_a \wedge (\neg x_b \vee x_d)$ en figure 5.12.

5.4 Conclusion

La caractérisation de la dynamique des réseaux d'automates est une tâche difficile. Dans ce chapitre, nous avons introduit le formalisme de modules, qui sont une généralisation des réseaux d'automates dont la dynamique ne peut être qu'encore plus complexe. Il est cependant important de ne pas percevoir les modules comme une simple généralisation des réseaux d'automates ; mais plutôt comme une opportunité de caractériser le comportement de ces derniers lorsque l'on considère non pas un réseau dans son ensemble, mais un morceau particulier qui est sujet à l'influence du reste. Dans ce cadre, les entrées d'un module permettent de simplifier son étude.

Malgré cette approche, l'étude des modules reste un sujet complexe, et il semble que la continuation de cette étude repose avant tout sur une restriction à des familles simples de modules. Un exemple d'une telle restriction est développée au chapitre suivant, qui est dédié à l'étude des modules acycliques.

6 Acyclicité

6.1 Les réseaux d'automates acycliques

Les réseaux d'automates acycliques sont une famille de réseaux d'automates parmi les plus simples, et consécutivement une des rares familles de réseaux d'automates dont nous comprenons entièrement le comportement. L'acyclicité ici est définie dans le contexte du graphe d'interaction du réseau ; un réseau d'automate est dit acyclique si son réseau d'interaction ne dispose d'aucun cycle.

Définition 64 (Réseau d'automates acyclique)**.** *Soit F un réseau d'automates. Le réseau F est dit* acyclique *si et seulement si son graphe d'interaction est un graphe acyclique.*

Exemple 44. *Soit F le réseau d'automates acyclique défini sur les automates $S = \{a, b, c\}$ et qui admet les fonctions locales suivantes :*

$$f_a(x) = 1$$
$$f_b(x) = \neg x_a$$
$$f_c(x) = x_a \vee x_b$$

Le graphe d'interaction du réseau d'automates acyclique F est représenté en figure 6.1, et le graphe de sa dynamique totale en figure 6.2.

Cela signifie plus formellement qu'il n'existe aucun chemin partant d'un automate s et retournant sur l'automate s dans le graphe d'interaction du réseau d'automates. Cela signifie qu'il existera toujours au moins un automate qui n'est influencé par aucun autre automate ; dans le cas contraire, on pourrait remonter l'influence dans le graphe sans fin, ce qui décrit naturellement un cycle. Ces automates sans aucune influence ont une fonction locale constante, puisqu'elle ne peut pas être influencée par aucun facteur, y compris la valeur de l'automate où elle est installée. Le reste du réseau prenant son influence toujours depuis ce sous-ensemble constant, et ne disposant d'aucun cycle, l'entièreté du réseau est destinée à se figer en un point fixe. Ce théorème est également évoqué en section 4.6.

Théorème 6 (Robert 1980)**.** *Soit F un réseau acyclique. La dynamique totale de F n'a qu'un seul attracteur qui est un point fixe.*

Démonstration. Soit F un réseau d'automates acyclique. On construit la séquence $(S_1, S_2, \ldots, S_k)$ telle que S_1 est l'ensemble des automates de F qui ne sont influencés

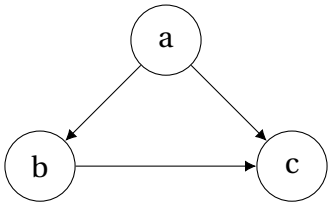

FIGURE 6.1 – Graphe d'interaction du réseau d'automates développé dans l'exemple 44.

par aucun autre automate, S_2 est l'ensemble des automates qui ne sont influencés que par des automates de S_1, et en général S_i décrit les automates qui ne sont influencés que par des sommets dans les ensembles qui le précèdent dans la séquence. En pratique, k décrit la longueur du plus long chemin dans le graphe d'interaction de F. Par définition, l'ensemble des éléments de cette séquence produit une partition de l'ensemble S, et aucun élément de cette séquence n'est vide.

On considère le graphe de la dynamique totale de F. Par définition, toute fonction locale d'un automate de S_1 est une fonction constante. Par conséquent, mettre à jour n'importe quel automate dans S_1 fige cet automate dans une valeur qui ne change plus. Plus formellement, toute configuration x qui vérifie $x_s \neq f_s(x)$ pour s dans S_1 ne fait pas partie d'un attracteur. Par conséquent, l'ensemble des attracteurs de la dynamique doit être inclus dans le sous-graphe dont les configurations vérifient $x_s = f_s(x)$ pour tout s dans S_1.

Nous considérons ce sous-graphe. Dans ce sous-graphe et par définition, toute fonction locale d'un automate de S_2 est une fonction qui ne dépend que d'automates dans S_1. Par définition de nouveau, le sous-graphe de la dynamique ne considère qu'une valeur constante pour chaque automate dans S_1. Par conséquent, le même raisonnement s'applique, et les attracteurs du réseau ne peuvent être que dans le sous-graphe de ce sous-graphe qui ne contient que les configurations qui vérifient $x_s = f_s(x)$ pour $s \in S_2$.

On applique ce raisonnement pour l'ensemble de la séquence $(S_1, S_2, \ldots, S_k)$, et obtenons un sous-graphe de la dynamique qui contient nécessairement l'ensemble des attracteurs de F. De plus, l'ensemble de ce raisonnement a selectionné les configurations qui figent l'ensemble des automates du réseau à une valeur constante. Par conséquent, le sous-graphe que nous obtenons ne possède qu'une seule configuration, qui est nécessairement le seul attracteur du graphe, et donc un point fixe. □

Ce résultat connu illustre parfaitement que la complexité des réseaux d'automates réside dans les cycles de son graphe d'interaction.

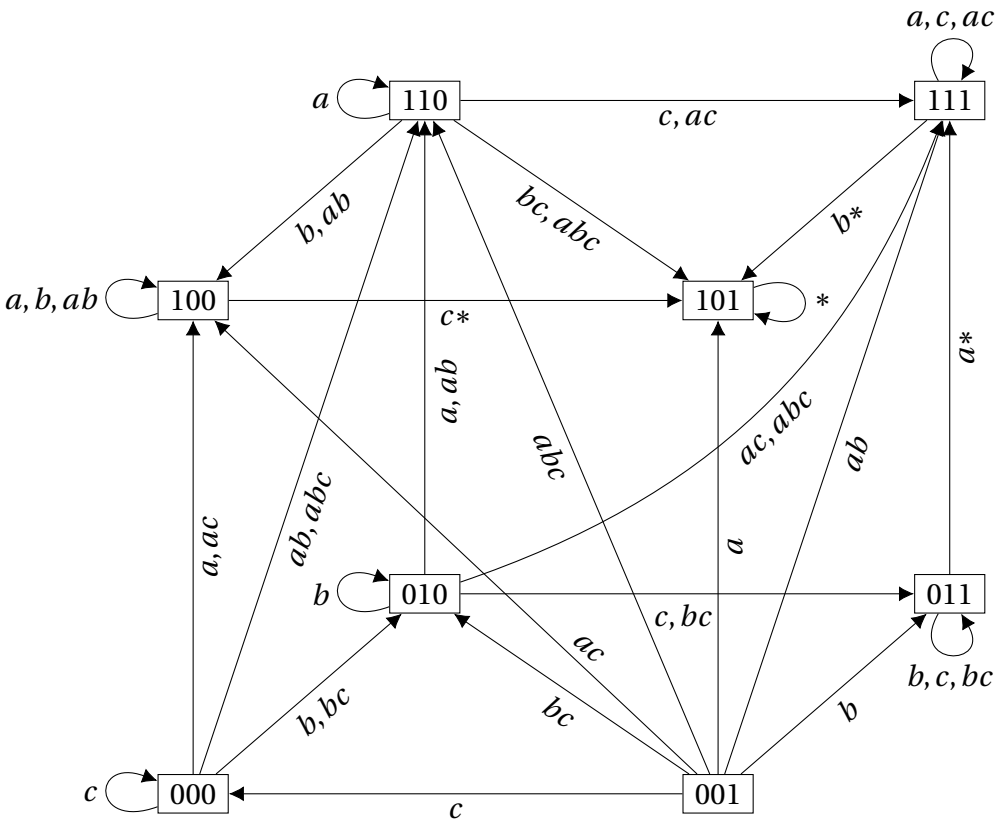

FIGURE 6.2 – Graphe de la dynamique totale du réseau d'automates développé dans l'exemple 44. L'étiquette ab représente la mise à jour $\delta = \{a, b\}$. L'étiquette $a*$ représente l'ensemble des mises à jour qui contiennent a, et $*$ l'ensemble de toutes les mises à jour possibles. On remarque que l'ensemble des attracteurs de cette dynamique se limite au point fixe 101.

6.2 Modules acycliques

Les modules sont une généralisation qui complexifie le comportement des réseaux étudiés. Nous développons dans cette section la restriction des modules sur des réseaux acycliques. Les réseaux d'automates acycliques étant des objets dont la dynamique est simple à comprendre, leur version modulaire dispose d'un comportement dynamique qui semble accessible à la caractérisation.

Définition 65 (Module acyclique)**.** *Soit M un module. Le réseau M est* acyclique *si et seulement si son graphe d'interaction est un graphe acyclique.*

Exemple 45. *Soit M le module acyclique défini sur les automates $S = \{a, b, c\}$, les entrées*

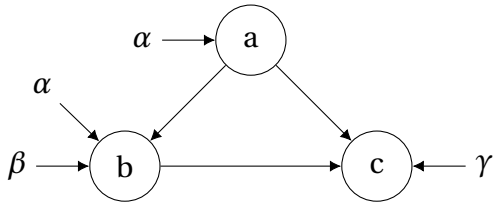

FIGURE 6.3 – Graphe d'interaction du module acyclique développé dans l'exemple 45.

$I = \{\alpha, \beta, \gamma\}$ et qui admet les fonctions locales suivantes :

$$f_a(x \cdot i) = i_\alpha$$
$$f_b(x \cdot i) = x_a \vee i_\beta \vee \neg i_\alpha$$
$$f_c(x \cdot i) = \neg x_b \wedge x_a \wedge \neg i_\gamma.$$

Le graphe d'interaction de M est représenté en figure 6.3.

Les modules acycliques diffèrent des réseaux d'automates acycliques car, en général, leurs entrées les empêchent de converger. Lorsque la valeur d'une entrée change dans une exécution, cette entrée provoque une cascade potentielle de réévaluations de fonctions locales le long du module. L'absence de cycle empêche le stockage de toute information ; et dès que les entrées sont figées sur une certaine configuration d'entrée, le réseau entier reprend le fonctionnement d'un réseau d'automates acyclique, et atteint un point fixe.

L'étude des modules acycliques nous semble justifiée par la simple réalisation que tout réseau d'automates, et même tout module, peut être exprimé comme le branchement récursif d'un module acyclique. Ce dernier se construit facilement en trouvant un coupe-cycle de sommets (ou Feedback Vertex Set) du graphe d'interaction du réseau, c'est-à-dire un ensemble d'automates qui, lorsque l'on remplace leur influence par des entrées, confèrent un réseau d'interaction acyclique. Afin de conserver au mieux la structure du réseau initial, il est préférable de trouver un Minimal Feedback Vertex Set du graphe d'interaction, ce qui est un problème NP-complet (KARP 1972).

Propriété 14. *Soit M un module. Il existe un module acyclique M' et un branchement récursif $\mathtt{w}$ tel que*

$$\circlearrowleft_{\mathtt{w}} M' = M.$$

Démonstration. Soit M un module. Nous considérons une solution du problème Minimal Feedback Vertex Set du graphe d'interaction de M que nous dénotons comme l'ensemble d'arêtes orientées $A \subseteq S \times S$. Par définition, retirer ces arêtes au graphe d'interaction de M nous donne un graphe acyclique.

On considère l'ensemble des automates $S' \subseteq S$, défini tel que $s' \in S'$ si et seulement s'il existe s' tel que $(s', s) \in A$. On définit un ensemble de variables $I' = \{\alpha'_s \mid s' \in S'\}$ tel

que $I \cap I' = \emptyset$.

Soit M' le module basé sur l'ensemble d'automates S et l'ensemble d'entrées $I \cup I'$, et qui, pour tout $s \in S$, définit la fonction locale f'_s comme la copie de la fonction f_s dans laquelle chaque variable x'_s est remplacée par $i_{\alpha_{s'}}$, pour tout s' dans S'. Soit $\mathtt{w}$ la fonction de branchement avec domaine I' qui définit $\mathtt{w}(\alpha_{s'}) = s'$. Ces constructions impliquent $\circlearrowleft_{\mathtt{w}} M' = M$. Par définition, le graphe d'interaction de M' est acyclique. □

Il semble clair qu'une caractérisation du comportement des modules acycliques permet une compréhension plus approfondie du fonctionnement des modules en général, et en particulier des réseaux d'automates.

6.2.1 Fonctions de sortie

Le comportement d'un automate d'un module acyclique peut être caractérisé comme une fonction de l'historique récent des entrées données au réseau. Ainsi, lorsqu'un module acyclique est mis à jour pour assez longtemps à partir d'une configuration initiale et avec une séquence de configurations d'entrée, l'influence de la configuration initiale sur l'évaluation des automates devient nulle. Cela se comprend clairement lorsque l'on se rappelle qu'un module acyclique ne dispose d'aucun mécanisme pour retenir de l'information. Nous caractérisons cette propriété par l'association d'une fonction de sortie à tout automate.

Ici, nous faisons une différence entre séquence d'entrée (composée par des configurations d'entrées mises dans un certain ordre) et historique d'entrée (composé de configurations d'entrées organisées par ancienneté). Si l'ensemble des séquences d'entrées d'un module est $(\Lambda^I)^*$, les historiques d'entrées de longueur n se définissent dans l'ensemble $\Lambda^{I\times\{1,\dots,n\}}$.

Définition 66. *Pour tout séquence d'entrée J et tout entier n tel que $|J| \geq n$, on définit l'historique de taille n extrait de J comme l'historique noté $H_n(J)$, qui pour tout $\alpha \in I$ et $k \in \{1,\dots,n\}$ définit*

$$H_n(J)_{(\alpha,k)} = (J_{|J|-k+1})_\alpha.$$

Une fonction de sortie est définie sur $O : \Lambda^{I\times\{1,\dots,D\}} \to \Lambda$, pour D un entier suffisamment grand. L'historique définit dans $\Lambda^{I\times\{1,\dots,D\}}$ passé en entrée d'une fonction de sortie contient les configurations d'entrée récentes. Ainsi, pour α une variable de I, l'index $(\alpha,1)$, que nous notons α_1, de ce vecteur, correspond à la valeur de l'entrée α à la mise à jour actuelle. L'index α_2 représente la valeur de l'entrée α une mise à jour dans le passé, et en général α_k représente la valeur de l'entrée α $k-1$ mises à jour dans le passé. Une fonction de sortie est une fonction qui, pour tout historique de longueur D, retourne une valeur dans Λ.

Pour O une fonction de sortie donnée, la valeur k maximale telle qu'il existe $\alpha \in I$ qui vérifie que α_k a une influence sur l'évaluation de O s'appelle le délai de O. Nous notons ce délai par la lettre $\mathtt{d}$, et il correspond à la profondeur minimale de l'historique nécessaire pour calculer O. Ainsi l'évaluation de O sur une séquence d'entrée consiste

à en extraire un historique de longueur $\mathtt{d}$, et d'évaluer O sur cet historique. On note que si $\mathtt{d} < D$, alors un historique de taille $\mathtt{d}$ ne contient pas assez d'information pour permettre un appel correct de la fonction O. Cependant, puisque par définition aucune variable qui manque à cet historique n'a d'influence sur le calcul de O, ne prenons le raccourcis de noter $O(H_{\mathtt{d}}(J))$ comme une évaluation correcte de O.

Définition 67 (Évaluation d'une fonction de sortie)**.** *Soit $O : \Lambda^{I\times\{1,\dots,D\}} \to \Lambda$ une fonction de sortie avec délai $\mathtt{d}$, et J une séquence d'entrée de taille $k \geq \mathtt{d}$. On définit l'*évaluation *de O sur J, notée $O(J)$, comme*

$$O(J) = O(H_{\mathtt{d}}(J)).$$

Nous définissons maintenant l'incrémentation d'une fonction de sortie, qui consiste à déplacer la fenêtre des variables considérées par la fonction dans le temps, ce qui a l'effet immédiat d'augmenter son délai de 1. Cette notion simple est centrale dans les preuves qui reposent sur le calcul de ces fonctions de sortie dans le contexte de la prédiction de la valeur d'automates dans un module acyclique.

Définition 68 (Incrémentation de fonction de sortie)**.** *Pour O une fonction de sortie de délai $\mathtt{d}$, nous définissons l'*incrémentation *de la fonction O, notée O^{+1}, comme la fonction de délai $\mathtt{d}+1$ définie telle que, pour toute séquence d'entrée J de taille k telle que $k \geq \mathtt{d}+1$,*

$$O^{+1}(J) = O(J_{[1,\dots,k-1]}).$$

Notre caractérisation du comportement sous mode de mise à jour parallèle des modules acycliques consiste à énoncer que, partant d'une configuration x, et pour une séquence d'entrée suffisament longue J, le comportement de tout automate peut être prédit par une fonction de sortie, dont l'évaluation ne dépend que de J et non de x.

Propriété 15. *Soit M un module acyclique. Pour s un automate, et J une séquence de configurations d'entrée suffisament longue de taille k, il existe au moins une fonction de sortie O_s de délai $\mathtt{d} \leq k$ telle que*

$$M(x,J)_s = O_s(J).$$

Démonstration. Dans cette preuve nous supposons disposer des fonctions locales du module M sous la forme de formules.

Soit prof(s) la fonction qui donne la profondeur d'un automate dans le graphe d'interaction définie telle que prof(s) = 1 si et seulement si s n'est influencé par aucun automate, et prof(s) = $\max_{s'\in \mathrm{prec}(s)} \mathrm{prof}(s') + 1$ pour prec(s) l'ensemble des automates qui influencent s dans le cas contraire.

Soit s tel que prof(s) = 1. Nous construisons O_s comme la copie de la fonction locale f_s dans laquelle chaque variable α est substituée par la variable α_1, pour tout α dans

I. Nous pouvons vérifier que pour toute séquence J de taille k au moins supérieure à 0, et toute configuration x, que $M(x,J)_s = O_s(J)$.

Supposons que, pour un certain entier i, nous disposons d'une fonction de sortie O_s pour chaque entier s qui vérifie $\text{prof}(s) = i$ telle que, pour toute séquence J de taille au moins i, $M(x,J)_s = O_s(J)$. Dans la suite de cette preuve, nous montrons que cette hypothèse implique la même proposition pour l'entier $i+1$, ce qui implique le résultat par induction.

Soit s un entier tel que $\text{prof}(s) = i+1$. Nous construisons la fonction de sortie O_s comme la copie de la fonction locale f_s dans laquelle chaque variable $x_{s'}$ est substituée par la fonction $O^{+1}_{s'}$, et chaque variable α par α_1. Soit J une séquence d'entrée de taille k au moins $i+1$. Nous observons que

$$M(x,J)_s = M(M(x,J_{[1,\dots,k-1]}),J_{[k]})_s$$
$$M(x,J)_s = f_s(M(x,J_{[1,\dots,k-1]}) \cdot J_k).$$

Cependant, par définition la fonction f_s n'est influencée que par des automates dont les fonctions de sorties sont déjà définies. Soit R l'ensemble des automates dont la profondeur est inférieure à s, et T l'ensemble des automates dont la profondeur est supérieure ou égale à s. Soit $O_R(J)$ la configuration définie sur R qui pour tout $r \in R$ définit la valeur $O_r(J)$. Pour toute configuration x_T sur T, nous observons que

$$M(x,J)_s = f_s(O_R(J_{[1,\dots,k-1]}) \cdot x_T \cdot J_k)$$
$$M(x,J)_s = O_s(J),$$

ce qui implique le résultat. □

L'ensemble des fonctions de sortie qui prédisent la valeur de s sont toutes très similaires. En effet, il ne s'agit que des variantes de la même fonction qui décalent leur analyse de l'historique dans le temps d'une valeur arbitraire, entraînant des délais plus grands. Plus formellement, chacune de ces différentes fonctions sont obtenues comme les incrémentations successives d'une fonction de sortie à délai minimal. Nous proposons donc la définition naturelle de fonction de sortie canonique associée à un automate comme la fonction de sortie qui prédit la valeur de l'automate, tout en ayant le délai minimal.

Définition 69 (Fonction de sortie canonique)**.** *Soit M un module acyclique et s un automate de M. La* fonction de sortie canonique *de s, notée O_s, est la fonction de sortie qui vérifie la propriété 15, avec délai minimal.*

Exemple 46. *Soit $S = \{a,b,c\}$ un ensemble d'automates. Soit M le module qui définit les fonctions locales suivantes :*

$$f_a(x \cdot i) = i_\alpha$$
$$f_b(x \cdot i) = x_a \vee i_\beta$$

$$f_c(x \cdot i) = x_b \wedge \neg x_a$$

Ce module vérifie les fonctions de sortie canoniques suivantes :

$$\begin{aligned} O_a &= \alpha_1 \\ O_b &= \alpha_2 \vee \neg\beta_1 \\ O_c &= (\alpha_3 \vee \neg\beta_2) \wedge \neg\alpha_2 \end{aligned}$$

Considérons la séquence d'entrée $J = (i, i', i'')$, avec $i_\alpha = i_\beta = i''_\alpha = 0$, et $i'_\alpha = i'_\beta = i''_\beta = 1$. Pour prédire la valeur de a, b ou c après que la séquence d'entrée J soit appliquée au module M, on calcule $O_a(J)$, $O_b(J)$, et $O_c(J)$. Pour cela, on doit transformer J, qui est une séquence d'entrée, en un historique des valeurs de α et β défini sur $\Lambda^{\{\alpha,\beta\}\times\{1,2,3\}}$. Ici, la longueur de notre séquence est $k = 3$, et nos fonctions de sorties ont chacune un délai que nous notons $\mathtt{d}_a = 1$, $\mathtt{d}_b = 2$ et $\mathtt{d}_c = 3$. En appliquant la définition 67, on obtient que $O_a(J) = O_a(H_1(J))$, $O_b(J) = O_b(H_2(J))$ et $O_c(J) = O_c(H_3(J))$. Les historiques $H_1(J)$, $H_2(J)$ et $H_3(J)$ sont tous issus de J, et tous de longueur différentes. On peut les évaluer de cette façon :

$$\begin{aligned} \forall k \in \{1,2,3\}, H_k(J)_{(\alpha,1)} &= (J_{|J|-1+1})_\alpha = i''_\alpha \\ \forall k \in \{1,2,3\}, H_k(J)_{(\beta,1)} &= (J_{|J|-1+1})_\beta = i''_\beta \\ \forall k \in \{2,3\}, H_k(J)_{(\alpha,2)} &= (J_{|J|-2+1})_\alpha = i'_\alpha \\ \forall k \in \{2,3\}, H_k(J)_{(\beta,2)} &= (J_{|J|-2+1})_\beta = i'_\beta \\ H_3(J)_{(\alpha,3)} &= (J_{|J|-3+1})_\alpha = i_\alpha \\ H_3(J)_{(\beta,3)} &= (J_{|J|-3+1})_\beta = i_\beta. \end{aligned}$$

Pour obtenir la valeur de $O_a(H_1(J))$, sachant que $O_a = \alpha_1$, il suffit de prendre la valeur de α dans la première configuration de l'historique, ce qui est $H_1(J)_{(\alpha,1)} = i''_\alpha = 0$. On a donc $O_a(J) = O_a(H_1(J)) = H_1(J)_{(\alpha,1)} = i''_\alpha = 0$.

Similairement, pour calculer $O_b(H_2(J))$, sachant que $O_b = \alpha_2 \vee \neg\beta_1$, on note que α_2 est la valeur de α donnée par la seconde configuration de l'historique, et que β_1 est la valeur de β donnée dans la première configuration de l'historique. Ces valeurs sont $H_2(J)_{(\alpha,2)} = i'_\alpha = 1$ et $H_2(J)_{(\beta,1)} = i''_\beta = 1$, ce qui signifie que $O_b(J) = O_b(H_2(J)) = H_2(J)_{(\alpha,2)} \vee \neg H_2(J)_{(\beta,1)} = i'_\alpha \vee \neg i''_\beta = 1 \vee \neg 1 = 1$.

Enfin, pour calculer $O_c(H_3(J))$, en sachant que $O_c = (\alpha_3 \vee \neg\beta_2) \wedge \neg\alpha_2$, on calcule $\alpha_3 = H_3(J)_{(\alpha,3)} = i_\alpha = 0$, $\beta_2 = H_3(J)_{(\beta,2)} = i'_\beta = 1$ et $\alpha_2 = H_3(J)_{(\alpha,2)} = i'_\alpha = 1$. On a donc $O_c(J) = O_c(H_3(J)) = (H_3(J)_{(\alpha,3)} \vee \neg H_3(J)_{(\beta,2)}) \wedge H_3(J)_{(\alpha,2)} = (i_\alpha \vee \neg i'_\beta) \wedge \neg i'_\alpha = (0 \vee \neg 1) \wedge \neg 1 = 0$.

L'évaluation des fonctions de sortie O_b et O_c sur J est illustrée en figure 6.4.

Exemple 47. *Soit M le module développé dans l'exemple 45, dont les fonctions locales*

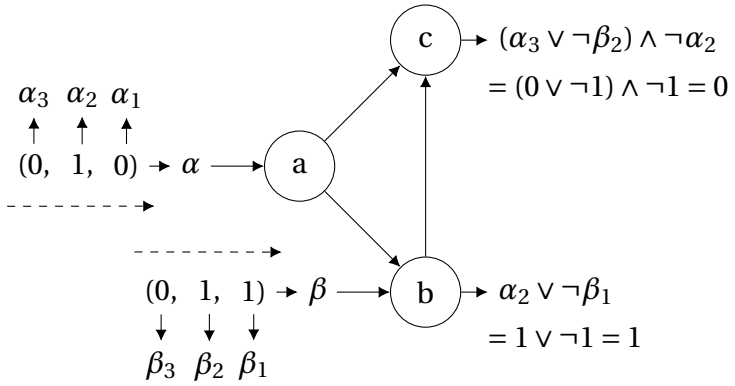

FIGURE 6.4 – Illustration du calcul opéré dans l'exemple 46. Sur la gauche, les séquences $(0,1,0)$ et $(0,1,1)$ sont les sous-séquences d'entrée extraites de J qui s'appliquent à α et β respectivement. Ces séquences se lisent de gauche à droite dans l'ordre de leur application sur le module. L'historique des entrées α et β est obtenu en inversant l'ordre de ces séquences : la variable α_1 dénote la plus récente valeur de α dans la séquence J, α_2 dénote la valeur qui précède α_1 dans le temps, et ainsi de suite. Sur la droite, on note les fonctions de sorties O_b et O_c sous forme de formules, ainsi que l'évaluation de ces formules sur la séquence d'entrée J.

sont

$$
\begin{aligned}
f_a(x \cdot i) &= i_\alpha \\
f_b(x \cdot i) &= x_a \vee i_\beta \vee \neg i_\alpha \\
f_c(x \cdot i) &= \neg x_b \wedge x_a \wedge \neg i_\gamma.
\end{aligned}
$$

Le module M vérifie les fonctions de sortie canoniques suivantes :

$$
\begin{aligned}
O_a &= \alpha_1 \\
O_b &= \alpha_2 \vee \beta_1 \vee \neg\alpha_1 \\
O_c &= \neg\alpha_3 \wedge \neg\beta_2 \wedge \alpha_2 \wedge \alpha_2 \wedge \neg\gamma_1.
\end{aligned}
$$

Les fonctions O_a, O_b et O_c ont délai 1, 2 et 3 respectivement.

6.2.2 Calcul d'une fonction de sortie booléenne

Pour M un module acyclique, nous nous intéressons au problème de calculer efficacement les fonctions de sortie canoniques des automates de M. La méthode que nous explorons est récursive, car les fonctions de sortie des automates plus profonds dans le graphe d'interaction de M dépendent directement de l'incrémentation des fonctions de sortie des automates à plus faible profondeur dans le graphe d'interaction. La

question centrale de l'efficacité de notre méthode repose sur le choix de l'encodage de ces fonctions de sortie.

En effet, pour n'importe laquelle de ces méthodes d'encodage, nous pouvons décrire un algorithme très général qui calcule les fonctions de sortie d'un module acyclique. Cette méthode repose sur les fonctions locales du module étudié, dans lesquelles les variables d'entrées sont substituées par les variables correspondantes de profondeur 1. Les variables d'automates sont quant à elles remplacées par l'incrémentation de la fonction de sortie de l'automate correspondant. Ce processus est bien défini car la structure acyclique du module abolit tout cycle d'influence.

Exemple 48. *Soit k un entier. On définit M_k le module acyclique constitué des automates $S = \{1, \dots, k\}$, et des entrées $\{\alpha^0, \alpha^1, \dots, \alpha^k\}$, tel que*

$$f_1(x \cdot i) = i_{\alpha^0} \oplus i_{\alpha^1}$$
$$\forall 1 < i \leq k, f_i(x \cdot i) = x_{i-1} \oplus i_{\alpha^i}.$$

Pour $k \in \mathbb{N}$, le module M_k développé dans l'exemple 48 est de taille une fonction linéaire en k. Considérons le calcul des fonctions de sortie de M_k sous la forme de formules booléennes avec les portes logiques $\{\wedge, \vee, \neg\}$. Par la procédure décrite précédemment, nous pouvons définir les fonctions de sortie du module de la façon suivante :

$$O_1(J) = (\alpha_1^0 \vee \alpha_1^1) \wedge (\neg\alpha_1^0 \wedge \neg\alpha_1^1)$$
$$\forall 1 < i \leq k, O_k(J) = (O_{i-1}^{+1} \vee \alpha_1^i) \wedge (\neg O_{i-1}^{+1} \vee \neg\alpha_1^i).$$

On note que la longueur du développement naïf de la formule booléenne qui encode la fonction de sortie O_i est deux fois supérieure à celle de la formule qui encode la fonction de sortie O_{i-1}. En général, cela signifie qu'il est possible que le coût du calcul de la fonction de sortie O_k soit de l'ordre de 2^k.

La même fonction de sortie possède une taille toujours polynomiale en la taille du module acyclique lorsque l'on considère l'encodage des fonctions de sortie du module sous la forme d'un circuit booléen. La différence provoquée par ce choix est que les multiples références à des fonctions de sortie précédentes n'explosent pas la taille du circuit obtenu grâce à la structure plus permissive des circuits booléens par rapport aux formules booléennes.

Propriété 16 (Perrot, Perrotin et Sené 2020b)**.** *Soit M un module acyclique, G son graphe d'interaction, et s un automate de M. Il existe un circuit booléen C calculable en temps polynomial en la taille de M et G qui encode la fonction de sortie O_s.*

Démonstration. Nous développons une méthode pour calculer en temps polynomial le circuit booléen qui encode la fonction de sortie de tout automate s dans M. Cette méthode calcule d'abord une liste de besoins. Cette liste est construite inductivement,

et vaut initialement $R_0 = \{(s,0)\}$, ce qui s'interprète par le besoin de construire la fonction de sortie de l'automate s avec le délai supplémentaire 0.

Nous construisons récursivement la liste de la façon suivante : $(t', \mathtt{d}+1) \in R_{k+1}$ si et seulement s'il existe $(t, \mathtt{d}) \in R_k$ tel que t' influence t dans le module M. La liste finale obtenue par cette méthode est définie par $R = \bigcup_i R_i$.

Proposition 1. *La liste R est calculable en temps polynomial en la taille de M.*

Afin de vérifier cette proposition, considérons D la longueur du plus long chemin dans M. Premièrement, il est facile de voir que, pour tout $k > D$, notre définition vérifie $R_k = \emptyset$. Ensuite, pour tout entier $k \leq D$, la valeur $\mathtt{d}$ maximale qui vérifie $(t, \mathtt{d}) \in R_k$ pour un certain automate t vérifie également $\mathtt{d} \leq k$. Nous pouvons donc conclure que la liste R_k est toujours de taille $n \times k$ au pire, pour n le nombre d'automates de M, ce qui est toujours moins que $n \times D$. Nous concluons donc que la liste totale est au maximum de taille $n \times D^2$, ce qui est polynomial. Notre méthode de construction recursive permettra donc de la calculer en temps polynomial, ce qui vérifie la Proposition 1.

Nous construisons le circuit C à partir de R de la façon suivante : nous prenons une instance du circuit qui encode la fonction locale f_t pour chaque paire $(t, \mathtt{d}) \in R$. Nous combinons l'ensemble des circuits obtenus en prenant toute variable du circuit étiquetée par i_α, pour $\alpha \in I$, et en la remplaçant par son équivalent $\alpha_{\mathtt{d}+1}$, pour $\mathtt{d}$ le délai ajouté du circuit actuellement modifié. Par exemple, pour $(t, \mathtt{d}) \in R$, tel que t dépend d'une entrée $\alpha \in I$, nous ajoutons au circuit C une copie du circuit qui encode f_t dans lequel toute occurence de la variable i_α a été remplacée par $\alpha_{\mathtt{d}+1}$. Enfin, nous connectons l'ensemble des circuits générés de cette façon en remplaçant toute occurence de la variable $x_{t'}$ dans le circuit de la paire $(t, \mathtt{d})$ par le littéral qui correspond à la porte de sortie du circuit qui encode l'automate t' avec le délai ajouté $\mathtt{d}+1$. Par définition de R, ce circuit fait toujours partie de notre assemblage. Le circuit total obtenu calcule la fonction de sortie O_s, et l'algorithme qui le calcule est au pire en $\mathcal{O}(n \times D^2 \times K)$, pour K la taille maximale d'encodage des circuits booléens donnés en entrée. □

6.2.3 Dynamique des modules acycliques

Les modules acycliques sont une restriction des modules dont le comportement, sous mise à jour parallèle, est caractérisé comme une fonction des séquences d'entrée utilisées pour les mettre à jour. Cette compréhension peut nous aider à mieux comprendre la dynamique des réseaux qui sont obtenus par branchement récursif de ces modules acycliques.

Considérons un module acyclique M, et F un réseau d'automates obtenu par le branchement récursif de M. Le nombre d'entrées de M nous permet une observation simple sur le nombre d'attracteurs contenus dans la dynamique parallèle de F. Par exemple, considérons les points fixes de F. Tout point fixe de F peut également être exprimé comme un point fixe de M, à condition que les entrées de M soient figées sur les valeurs qui correspondent aux automates dont l'influence remplace ces entrées

pour obtenir F. Autrement dit, tout point fixe, et en général tout attracteur de F, peut être observé sur M à condition de sélectionner la séquence d'entrée qui mime le branchement qui construit F à partir de M. Ce simple fait est une observation directe de la priopriété 12.

Dans le cas acyclique, ce cas implique que nous pouvons définir une borne supérieure sur le nombre d'attracteurs de F, par le nombre d'attracteurs similaires que l'on peut observer sur M en supposant une séquence d'entrée qui fonctionne en ce sens. Cette dernière valeur est bornée par le nombre d'entrées de M. En effet, lorsque les valeurs des entrées de M sont figées pour reproduire un point fixe, l'ensemble des états du reste du réseau est une fonction de ces entrées. Cela signifie que le nombre de points fixes de F est borné par le nombre de configurations d'entrée possibles de M. Cet argument, porté pour toute taille d'attracteur, nous permet d'obtenir le résultat suivant.

Théorème 7 (PERROT, PERROTIN et SENÉ 2020a)**.** *Soit M un module acyclique avec k entrées, et $\mathtt{w}$ une fonction de branchement récursif telle que $\circlearrowleft_{\mathtt{w}} M$ est un réseau d'automates. Notons $a(k,c)$ le nombre d'attracteurs de taille c dans la dynamique parallèle de $\circlearrowleft_{\mathtt{w}} M$. Nous énonçons que $a(k,c) \leq A(k,c)$, pour :*

$$A(k,1) = |\Lambda|^k$$
$$A(k,c) = |\Lambda|^{kc} - \sum_{c'<c, c' \mid c} A(k,c').$$

Démonstration. Soit M un module avec k entrées, et $\mathtt{w}$ une fonction de branchement qui vérifie $\mathrm{dom}(\mathtt{w}) = I$. Soit $\mathtt{w}(I)$ la notation de l'ensemble $\mathtt{w}(I) = \{\mathtt{w}(\alpha) \mid \alpha \in I\}$. Nous observons le fait suivant :

$$|\mathtt{w}(I)| \leq |I| = k. \tag{6.1}$$

Considérons maintenant le réseau d'automates $F = \circlearrowleft_{\mathtt{w}} M$, et $X = (x_1, x_2, \ldots, x_c)$ un attracteur de sa dynamique. Par définition, X vérifie $F(x_i) = x_{i+1}$ pour $i < c$, $F(x_c) = x_1$ et pour tout i, j, $x_i = x_j \implies i = j$. Pour $R \subseteq S$, nous définissons $X|_R$ la séquence qui vérifie $X|_R = (x_1|_R, x_2|_R, \ldots, x_c|_R)$. Supposons X' un autre attracteur de taille c. Nous faisons la proposition suivante.

Proposition 2. $X|_{\mathtt{w}(I)} = X'|_{\mathtt{w}(I)} \implies X = X'$.

Pour vérifier cette proposition, supposons que $X|_{\mathtt{w}(I)} = X'|_{\mathtt{w}(I)}$. Par hypothèse, M est acyclique, et il existe un ensemble non vide $S_1 \subseteq S$ qui contient exactement tous les automates de M qui ne sont influencés que par des entrées. Cela signifie que notre hypothèse implique $X|_{\mathtt{w}(I) \cup S_1} = X'|_{\mathtt{w}(I) \cup S_1}$. Maintenant, considérons S_2 l'ensemble non vide qui contient exactement tous les automates influencés seulement par des entrées et des automates dans S_1. Cela nous permet de déduire que $X|_{\mathtt{w}(I) \cup S_1 \cup S_2} = X'|_{\mathtt{w}(I) \cup S_1 \cup S_2}$. En construisant l'ensemble S_i composé exactement des automates influencés par des entrées et des automates dans les ensembles qui précèdent, et sachant qu'aucun de ces ensembles n'est vide, nous prouvons la proposition par induction.

Cette proposition nous permet de conclure qu'il ne peut y avoir dans la dynamique de F plus d'attracteurs de taille c que d'ensembles $X|_{\mathtt{w}(I)}$ distincts. Plus formellement,

$$a(k,c) \leq |\Lambda|^{kc}. \tag{6.2}$$

Soit X une des possibles $|\Lambda|^{kc}$ différentes séquences de c configurations. Supposons que $F(x_i) = x_{i+1}$ pour $i < c$ et $F(x_c) = x_1$. Par définition, s'il existe i, j tels que $i \neq j$ et $x_i = x_j$, alors X est une séquence périodique, et n'est pas un attracteur. Cela est équivalent à trouver une séquence X' plus petite telle que X peut être composé comme la répétition X'^q, pour q un entier positif. Cela implique que, pour toute séquence X' telle que la taille de X' divise c, il existe une séquence $X = X'^q$ pour q un entier positif tel que X est compté par la formule $|\Lambda|^{kc}$, et X n'est pas un attracteur. Nous pouvons donc raffiner la borne précédente et affirmer que $a(k,c)$ n'est pas plus grand que $|\Lambda|^{kc} - \sum_{x'<c, c'|c} A(k,c')$. □

De ce théorème central, nous pouvons établir un ensemble de résultats simples qui permettent d'illustrer que la source principale de la complexité des modules acycliques dans le cas booléen est le nombre de leurs entrées, et non le nombre de leurs automates. Pour ce faire, nous décrivons les problèmes de décision suivant.

▸ ACYCLIC MODULE ATTRACTOR PROBLEM

Entrée: Un module acyclique booléen M, k entrées et n automates, une fonction de branchement $\mathtt{w}$ et un nombre c encodé en unaire.

Promesse: L'encodage des fonctions locales de M ne comporte que des variables essentielles.

Question: Existe-t-il un attracteur de taille c dans la dynamique parallèle du réseau d'automates $\circlearrowleft_{\mathtt{w}} M$?

▸ ACYCLIC MODULE FIXED POINT PROBLEM

Entrée: Un module acyclique booléen M, k entrées et n automates et une fonction de branchement $\mathtt{w}$.

Promesse: L'encodage des fonctions locales de M ne comporte que des variables essentielles.

Question: Existe-t-il une configuration x telle que $\circlearrowleft_{\mathtt{w}} M(x) = x$?

Ces problèmes possèdent les complexités suivantes.

Théorème 8 (PERROT, PERROTIN et SENÉ 2020a)**.** *Le problème Acyclic Module Attractor peut être résolu en temps $\mathcal{O}(f(k \times c)q(n))$ pour f une fonction et q un polynôme,* i.e. *il est dans FPT.*

Démonstration. L'algorithme qui prouve ce théorème vérifie toutes les séquences d'entrées pour k entrées et de longueur c possibles, dont il existe $|\mathbb{B}|^{k\times c}$ représentantes au total. Pour vérifier si une telle séquence J décrit un attracteur de taille c dans la dynamique du réseau $\circlearrowleft_{\mathtt{w}} M$, il suffit de mettre à jour le module M avec une séquence d'entrées composée comme la répétition de la séquence J jusqu'à ce que sa taille

dépasse le délai maximal parmi les fonctions de sortie de M. Un attracteur sera obtenu si et seulement si la séquence récurrente obtenue en sortie est identique à la séquence J, à l'application de la fonction `w` près. Cette procédure s'exécute en temps $r(n \times k \times c)$, pour r un polynôme.

Nous observons que tout attracteur de taille c dans la dynamique de $\circlearrowleft_{\mathtt{w}} M$ correspond nécessairement à une séquence d'entrée J, qui peut être récupérée simplement par la lecture des automates branchés par la fonction `w` pendant c étapes. Notre procédure est donc fiable.

En testant chaque séquence J, on établit un algorithme qui vérifie l'existence d'un attracteur de taille c en une complexité $\mathcal{O}(|\mathbb{B}|^{k\times c} \times r(n \times k \times c)$, ce qui signifie qu'il existe une fonction f et un polynôme q tels que la complexité de cet algorithme est $\mathcal{O}(f(k \times c)q(n))$. □

Théorème 9 (PERROT, PERROTIN et SENÉ 2020a)**.** *Le problème Acyclic Module Fixed Point est NP-complet.*

Démonstration. Tout d'abord, ce problème est dans NP car, pour toute configuration, vérifier qu'il s'agit d'un point fixe demande simplement d'exécuter le module une fois.

Pour démontrer qu'il s'agit d'un problème NP-difficile, on procédons à une réduction depuis le problème SAT. Pour f une formule, on construit un module qui dispose d'un automate pour chaque variable dans f, ainsi que deux autres automates `solver` et `oscillator`. Chaque automate correspondant à une variable dispose d'une entrée unique qui sont toutes branchées par `w` à l'automate qui la contient ; de façon à ce qu'une fois branchés, chacun de ces automates a dispose d'une fonction locale triviale $f_a(x) = x_a$.

La fonction locale de l'automate `solver` est une réécriture de la formule f dans laquelle chaque variable est remplacée par l'influence de l'automate qui lui correspond. Enfin, la fonction locale de l'automate `oscillator` est définie comme $f_{\mathtt{oscillator}}(x) = \neg\mathtt{solver} \wedge \neg\mathtt{oscillator}$; cet automate oscille entre les valeurs 0 et 1, excepté lorsque l'automate `solver` est évalué à 1, alors la valeur de `oscillator` se fixe à 0.

Par construction, l'évaluation de chaque automate qui n'est pas `solver` ni `oscillator` ne change pas dans le temps. Cela implique que la valeur de l'automate `solver` se fixe après une mise à jour. On en conclut que la valeur du dernier automate, `oscillator`, va se fixer, et le réseau va obtenir un point fixe si et seulement si la formule f est satisfaite ; on en déduit la réduction, et le résultat. □

6.2.4 Caractérisation de la dynamique par les fonctions de sortie

Les fonctions de sortie d'un module acyclique M permettent l'exécution complète du module M en ne prenant en compte que les dernières d configurations dans la séquence de configurations d'entrée, pour d le délai maximal parmi les fonctions de sortie de M. Nous établissons que les fonctions de sortie de M caractérisent son

comportement asymptotique. Il est en effet intéressant de noter que les fonctions de sortie d'un module acyclique ne permettent pas de prédire l'évaluation d'un automate sur de courtes exécutions. Cela s'explique par l'observation que la configuration initiale d'une module, pour une exécution, possède une influence sur le calcul de ses automates, jusqu'à un nombre de mise à jour égal à la profondeur maximale du réseau, point auquel le comportement de tous les automates du module devient une fonction de l'historique de ses entrées.

Cette idée de caractérisation du comportement asymptotique d'un module par ses fonctions de sortie se raffine de façon concrète par le théorème suivant. Celui-ci énonce que, si deux modules acycliques possèdent suffisament de fonctions de sortie en commun et sont branchés récursivement de façon similaire relativement à ces fonctions de sortie pour obtenir des réseaux d'automates, alors les attracteurs des dynamiques parallèles respectives de ces réseaux sont isomorphes.

Théorème 10 (Perrot, Perrotin et Sené 2020a)**.** *Soient M et M' deux modules acycliques, avec T et T' des sous-ensembles de leurs ensembles d'automates tels que $|T| = |T'|$. S'il existe g une bijection de I vers I' et h une bijection de T vers T' telle que pour tout s dans T, les fonctions O_s et $O'_{h(s)}$ ont même délai, et que pour toute séquence d'entrée J de longueur au moins le délai de O_s,*

$$O_s(J) = O'_{h(s)}(J \circ g^{-1})$$

alors, pour toute fonction de branchement $\mathtt{w} : I \to T$, les graphes de la dynamique parallèle des réseaux d'automates $\circlearrowleft_{\mathtt{w}} M$ et $\circlearrowleft_{h\circ\mathtt{w}\circ g^{-1}} M'$ ont des attracteurs isomorphes, au nommage des automates près.

Démonstration. En préliminaire, nous remarquons que $\mathtt{w}$ a pour domaine I, et donc que $\circlearrowleft_{\mathtt{w}} M$ est un réseau d'automates que nous noterons F. La définition de g nous permet la même remarque pour $\circlearrowleft_{h\circ\mathtt{w}\circ g^{-1}} M'$, que nous noterons F'. Soient G et G' les attracteurs des graphes de la dynamique parallèle de F et F' respectivement. L'objectif de cette preuve est de prouver que les sous-graphes G et G' sont isomorphes.

Pour toute configuration $x \in \Lambda^S$, nous définissons la séquence d'entrée de taille k générée par x, que nous notons $\hat{J}(x)^k$, comme la séquence qui vérifie

$$\hat{J}(x)^k_l = F^{l-1}(x)|_T \circ \mathtt{w}, \text{pour } 1 \leq l \leq k.$$

Intuitivement, la séquence $J(x)^k_l$ contient les valeurs successives des automates de T qui sont considérés comme les sorties du réseau F. Cet historique démarre à la configuration x et s'arrête après k mises à jour. La séquence obtenue est combinée avec le branchement $\mathtt{w}$, ce qui en fait une séquence d'entrée.

Proposition 3. *Soit k un entier qui vérifie $d_s \leq k$ pour tout automate s, et pour d_s le délai de la fonction de sortie canonique de s. Pour J une séquence d'entrée de longueur k, l'évaluation $M(x, J)$ ne dépend que de J, et non de la configuration de départ x.*

Cette proposition se vérifie immédiatement par l'application de la propriété 15. Le calcul d'une fonction de sortie ne dépend pas de la configuration initiale sur des exécutions suffisament longues. En nous fondant sur cette proposition, nous établissons la notation $M(x, J) = M(J)$ dans le reste de cette demonstration lorsqu'applicable, c'est-à-dire lorsque la séquence J est plus longue que le délai maximal parmi les fonctions de sortie de M.

Proposition 4. *Soit J une séquence d'entrée telle que $M(J)$ est bien définie. Nous énonçons que $\hat{J}(M(J)^k = J \implies M(J) \in V(G)$, pour $V(G)$ l'ensemble des sommets du graphe des attracteurs de F.*

Cette proposition énonce que si la configuration $M(J)$ génère la séquence d'entrée J lorsque utilisée pour mettre à jour le réseau F, alors $M(J)$ fait partie d'un attracteur. Pour le prouver, notons $x = M(J)$. Par hypothèse, $\hat{J}(x)^k = J$. Cela implique

$$\begin{aligned}
F^k(x) =& F(F^{k-1}(x)) = M(F^{k-1}(x), F^{k-1}(x)|_T \circ \mathtt{w}) \\
=& M(M(\dots M(x, F^0(x)|_T \circ \mathtt{w}) \dots, F^{k-2}(x)|_T \circ \mathtt{w}), F^{k-1}(x)|_T \circ \mathtt{w}) \\
=& M(M(\dots M(x, \hat{J}(x)_1^k) \dots, \hat{J}(x)_{k-1}^k), \hat{J}(x)_k^k) \\
=& M(x, \hat{J}(x)^k) = M(x, J) = M(J) = x.
\end{aligned}$$

En conclusion, $F^k(x) = x$, ce qui implique que x fait partie d'un attracteur de taille k, ce qui vérifie la proposition.

Proposition 5. *Soit x dans $V(G)$. Il existe x' dans $V(G')$ tel que $\hat{J}(x)^k \circ g^{-1} = \hat{J}(x')^k$, pour tout entier k.*

Cette proposition implique que pour toute configuration récursive de F, il existe une configuration récursive de F' qui génère une séquence d'entrées identique à l'application de la bijection g près.

Pour le prouver, considérons $x \in V(G)$ et k un entier supérieur au délai de toute fonction de sortie de M et M', tels que $F^k(x) = x$. Considérons également les séquences d'entrées $\hat{J}(x)^k$ et $\hat{J}(x)^k \circ g^{-1}$. La proposition 3 implique que $M'(\hat{J}(x)^k \circ g^{-1})$ est une configuration de M' bien définie, que nous notons x'. Nous devons maintenant prouver que $\hat{J}(x)^k \circ g^{-1} = \hat{J}(x')^k$. D'après nos définitions,

$$\hat{J}(x)_1^k \circ g^{-1} = F^0(x)|_T \circ \mathtt{w} \circ g^{-1} = x|_T \circ \mathtt{w} \circ g^{-1},$$

alors que

$$\begin{aligned}
\hat{J}(x')_1^k = F'^0(x')|_{T'} \circ h \circ \mathtt{w} \circ g^{-1} =& x'|_{T'} \circ h \circ \mathtt{w} \circ g^{-1} \\
=& M'(\hat{J}(x)^k \circ g^{-1})|_{T'} \circ h \circ \mathtt{w} \circ g^{-1}.
\end{aligned}$$

Nous observons que, pour tout s' dans T', $M'(\hat{J}(x)^k \circ g^{-1})_{s'} = O'_{s'}(\hat{J}(x)^k \circ g^{-1})$, ce qui,

par l'hypothèse du théorème, est égal à $O_{h^{-1}(s')}(\hat{J}(x)^k)$. Cela implique que

$$M'(\hat{J}(x)^k \circ g^{-1})|_{T'} \circ h = M(\hat{J}(x)^k)|_T \circ h^{-1} \circ h = x|_T$$

ce dont nous déduisons

$$\hat{J}(x')_1^k = x|_T \circ \mathtt{w} \circ g^{-1} = \hat{J}(x)_1^k \circ g^{-1}.$$

En conclusion, $\hat{J}(x)_1^k \circ g^{-1} = \hat{J}(x')_1^k$.

Ce fait marque la première étape d'une démonstration par induction pour prouver $\hat{J}(x)^k \circ g^{-1} = \hat{J}(x')^k$. Supposons donc que, pour un certain ℓ inférieur à k,

$$\hat{J}(x)_{[1,\ell]}^k \circ g^{-1} = \hat{J}(x')_{[1,\ell]}^k.$$

Montrons désormais que $\hat{J}(x)_{[1,\ell+1]}^k \circ g^{-1} = \hat{J}(x')_{[1,\ell+1]}^k$. Pour cela, il nous est simplement nécessaire de montrer que $\hat{J}(x)_{\ell+1}^k \circ g^{-1} = \hat{J}(x')_{\ell+1}^k$. Nous savons que

$$\begin{aligned}
\hat{J}(x)_{\ell+1}^k \circ g^{-1} &= F^\ell(x)|_T \circ \mathtt{w} \circ g^{-1} \\
&= M(x, \hat{J}(x)_{[1,\ell]}^k)|_T \circ \mathtt{w} \circ g^{-1} \\
&= M(M(\hat{J}(x)^k), \hat{J}(x)_{[1,\ell]}^k)|_T \circ \mathtt{w} \circ g^{-1} \\
&= M(\hat{J}(x)^k \cdot \hat{J}(x)_{[1,\ell]}^k)|_T \circ \mathtt{w} \circ g^{-1},
\end{aligned}$$

et

$$\begin{aligned}
\hat{J}(x')_{\ell+1}^k &= F'^\ell(x')|_{T'} \circ h \circ \mathtt{w} \circ g^{-1} \\
&= M'(x', \hat{J}(x')_{[1,\ell]}^k)|_{T'} \circ h \circ \mathtt{w} \circ g^{-1} \\
&= M'(x', \hat{J}(x)_{[1,\ell]}^k \circ g^{-1})|_{T'} \circ h \circ \mathtt{w} \circ g^{-1} \\
&= M'(M'(\hat{J}(x)^k \circ g^{-1}), \hat{J}(x)_{[1,\ell]}^k \circ g^{-1})|_{T'} \circ h \circ \mathtt{w} \circ g^{-1} \\
&= M'((\hat{J}(x)^k \circ g^{-1}) \cdot (\hat{J}(x)_{[1,\ell]}^k \circ g^{-1}))|_{T'} \circ h \circ \mathtt{w} \circ g^{-1} \\
&= M'((\hat{J}(x)^k \cdot \hat{J}(x)_{[1,\ell]}^k) \circ g^{-1})|_{T'} \circ h \circ \mathtt{w} \circ g^{-1}.
\end{aligned}$$

Nous observons que la séquence $(\hat{J}(x)^k \cdot \hat{J}(x)_{[1,\ell]}^k) \circ g^{-1})$ est de taille au moins k, ce qui nous permet de l'utiliser pour calculer le résultat d'une fonction de sortie. De l'hypothèse du théorème, nous déduisons que pour tout s' dans T',

$$\begin{aligned}
M'((\hat{J}(x)^k \cdot \hat{J}(x)_{[1,\ell]}^k) \circ g^{-1})_{s'} &= O'_{s'}((\hat{J}(x)^k \cdot \hat{J}(x)_{[1,\ell]}^k) \circ g^{-1}) \\
&= O_{h^{-1}(s')}(\hat{J}(x)^k \cdot \hat{J}(x)_{[1,\ell]}^k) \\
&= M(\hat{J}(x)^k \cdot \hat{J}(x)_{[1,\ell]}^k)_{h^{-1}(s')}
\end{aligned}$$

ce qui, par une autre utilisation de l'hypothèse du théorème, implique

$$\begin{aligned}\hat{J}(x')^k_{\ell+1} &= M'((\hat{J}(x)^k \cdot \hat{J}(x)^k_{[1,\ell]}) \circ g^{-1})|_{T'} \circ h \circ \mathtt{w} \circ g^{-1} \\ &= M(\hat{J}(x)^k \cdot \hat{J}(x)^k_{[1,\ell]})_T \circ h^{-1} \circ h \circ \mathtt{w} \circ g^{-1} \\ &= M(\hat{J}(x)^k \cdot \hat{J}(x)^k_{[1,\ell]})_T \circ \mathtt{w} \circ g^{-1} \\ &= \hat{J}(x)^k_{\ell+1} \circ g^{-1}\end{aligned}$$

ce qui termine notre preuve par induction, de laquelle nous déduisons que $\hat{J}(x)^k \circ g^{-1} = \hat{J}(x')^k$. Nous en déduisons $\hat{J}(M'(\hat{J}(x)^k \circ g^{-1}))^k = \hat{J}(x')^k = \hat{J}(x)^k \circ g^{-1}$, ce qui, à l'aide de la proposition 4, implique que x' fait partie de $V(G')$, et que x' fait partie d'un attracteur dont la taille divise k, tout comme x. Ceci conclut la preuve de la proposition 5 en supposant k suffisament grand, mais nous notons que la proposition se maintient pour tout entier k.

Nous observons une séquence symétrique d'arguments pour prouver que, pour toute configuration x' dans $V(G')$, il existe x dans $V(G)$ tel que $\hat{J}(x')^k \circ g = \hat{J}(x)^k$. De ceci, nous déduisons que, pour tout x dans $V(G)$, il existe une unique configuration x' dans $V(G')$ qui vérifie cette équation. Cela se voit par le fait que s'il existait x', x'' dans $V(G')$ telles que $\hat{J}(x')^k \circ g = \hat{J}(x)^k$, et $\hat{J}(x'')^k \circ g = \hat{J}(x)^k$, alors $\hat{J}(x')^k = \hat{J}(x'')^k$, ce qui implique pour k un multiple des tailles des attracteurs qui contiennent x' et x'' que $x' = M'(\hat{J}(x')^k) = M'(\hat{J}(x'')^k) = x''$, et x' x'' sont la même configuration.

Soit $\hat{h} : V(G) \to V(G')$ la bijection qui, à tout x dans $V(G)$, associe x' dans $V(G')$ telle que $\hat{J}(x')^k \circ g = \hat{J}(x)^k$, ce qui implique que $\hat{h}(x) = M'(\hat{J}(x)^k \circ g^{-1})$, pour k plus grand que le délai de toute fonction de sortie dans M et M' et multiple de la taille des attracteurs qui contiennent x et $\hat{h}(x)$. Le reste de cette preuve montre que $\hat{h}$ est un isomorphisme de G vers G', ce qui demande de montrer que, pour tout x dans $V(G)$, $\hat{h}(F(x)) = F'(\hat{h}(x))$.

Soit x dans $V(G)$ et k un entier multiple de la longueur de l'attracteur auquel x appartient, tel que k est plus grand que le délai de toutes les fonctions de sortie de M et M'. Nous savons que

$$\begin{aligned}\hat{h}(F(x)) &= M'(\hat{J}(F(x))^k \circ g^{-1}) \\ &= M'((F^0(F(x))|_T \circ \mathtt{w}, F^1(F(x))|_T \circ \mathtt{w}, \ldots, F^{k-1}(F(x))|_T \circ \mathtt{w}) \circ g^{-1}) \\ &= M'((F^1(x)|_T \circ \mathtt{w}, F^2(x)|_T \circ \mathtt{w}, \ldots, F^k(x)|_T \circ \mathtt{w}) \circ g^{-1}) \\ &= M'((F^1(x)|_T \circ \mathtt{w} \circ g^{-1}, F^2(x)|_T \circ \mathtt{w} \circ g^{-1}, \ldots, F^k(x)|_T \circ \mathtt{w} \circ g^{-1})).\end{aligned}$$

Considérons un élément individuel de la séquence çi-dessus, $F^\ell(x)|_T \circ \mathtt{w} \circ g^{-1}$. Pour tout s dans S,

$$\begin{aligned}F^\ell(x)_s &= M(x, \hat{J}(x)^\ell)_s \\ &= M(M(\hat{J}(x)^k), \hat{J}(x)^\ell)_s \\ &= M(\hat{J}(x)^k \cdot \hat{J}(x)^\ell)_s\end{aligned}$$

$$
\begin{aligned}
&= O_s(\hat{J}(x)^k \cdot \hat{J}(x)^\ell) \\
&= O'_{h(s)}((\hat{J}(x)^k \cdot \hat{J}(x)^\ell) \circ g^{-1}) \\
&= M'((\hat{J}(x)^k \cdot \hat{J}(x)^\ell) \circ g^{-1})_{h(s)} \\
&= M'(M'(\hat{J}(x)^k \circ g^{-1}), \hat{J}(x)^\ell \circ g^{-1})_{h(s)} \\
&= M'(\hat{h}(x), \hat{J}(x)^\ell \circ g^{-1})_{h(s)} \\
&= F'^\ell(\hat{h}(x))_{h(s)}
\end{aligned}
$$

ce qui implique que $F^\ell(x)|_T \circ \mathtt{w} \circ g^{-1} = F'^\ell(\hat{h}(x))|_{T'} \circ h \circ \mathtt{w} \circ g^{-1}$. En remplaçant ces termes dans l'équation initiale, nous obtenons

$$
\begin{aligned}
\hat{h}(F(x)) &= M'((F'^1(\hat{h}(x))|_{T'} \circ h \circ \mathtt{w} \circ g^{-1}, F'^2(\hat{h}(x))|_{T'} \circ h \circ \mathtt{w} \circ g^{-1}, \dots \\
&\qquad \dots, F'^k(\hat{h}(x))|_{T'} \circ h \circ \mathtt{w} \circ g^{-1})) \\
&= M'(\hat{J}(\, F'(\hat{h}(x)))) \\
&= F'(\hat{h}(x)),
\end{aligned}
$$

ce qui conclut cette démonstration. □

Exemple 49. *Soit M le module acyclique défini sur les automates $S = \{a, b, c, d\}$ et l'entrée $I = \{\alpha\}$, et qui admet les fonctions locales*

$$
\begin{aligned}
f_a(x \cdot i) &= i_\alpha \\
f_b(x \cdot i) &= x_a \vee i_\alpha \\
f_c(x \cdot i) &= \neg x_a \\
f_d(x \cdot i) &= (x_b \wedge \neg x_c) \vee i_\alpha.
\end{aligned}
$$

Nous observons les fonctions de sortie canoniques

$$
\begin{aligned}
O_a(J) &= \alpha_1 \\
O_b(J) &= \alpha_2 \vee \alpha_1 \\
O_c(J) &= \neg \alpha_2 \\
O_d(J) &= ((\alpha_3 \vee \alpha_2) \wedge \alpha_3) \vee \alpha_1 \\
&= \alpha 3 \vee \alpha_1.
\end{aligned}
$$

Soit M' le module acyclique défini sur les automates $S' = \{l, m, n\}$ et l'entrée $I' = \{\beta\}$ qui admet les fonctions locales

$$
\begin{aligned}
f'_l(x \cdot i) &= i_\alpha \\
f'_m(x \cdot i) &= x_l \\
f'_n(x \cdot i) &= x_m \vee i_\alpha.
\end{aligned}
$$

Nous observons les fonctions de sortie canoniques

$$O'_l(J) = \beta_1$$
$$O'_m(J) = \beta_2$$
$$O'_n(J) = \beta_3 \vee \beta_1.$$

Soient $T = \{d\}, T' = \{n\}$, $g : I \to I'$ la bijection qui définit $g(\alpha) = \beta$, et $h : T \to T'$ la bijection qui définit $h(d) = n$. Les modules M et M' vérifient $O_d(J) = O'_{h(d)}(J \circ g^{-1})$, et par conséquent, pour $\mathtt{w} : T \to I$ le branchement qui définit $\mathtt{w}(d) = \alpha$, les réseaux d'automates $\circlearrowleft_{\mathtt{w}} M$ et $\circlearrowleft_{h \circ \mathtt{w} \circ g^{-1}} M'$ ont des attracteurs isomorphes.

Ce théorème permet une notion nouvelle d'équivalence sur les modules acycliques pour lesquels on désigne un sous-ensemble d'automates T comme sorties. Cette notion d'équivalence est vérifiée pour toute paire de modules pour lesquels les conditions nécessaires sont vérifiées pour que le théorème 10 s'applique. Afin d'explorer ces classes d'équivalence de façon productive, nous définissons une restriction sur les ensembles T définissables pour un module M. Cette restriction n'accepte que les modules qui ne possèdent pas de branches inutiles, à savoir ceux dont tous les automates influencent directement ou indirectement un automate dans T.

Définition 70 (Ensemble de sortie)**.** *Soit M un module acyclique, et T un sous-ensemble de S. On définit que T est un* ensemble de sortie *de M si et seulement si, pour tout automate s, il existe un chemin dans le graphe d'interaction de M depuis s vers t, pour t un automate dans T.*

Naturellement, cette définition restreint notre étude à des ensembles de sorties non vides. Nous pouvons maintenant définir les classes d'équivalence sur les paires (M, T), pour M un module acyclique et T un ensemble de sortie de M.

Définition 71 (Équivalence en sortie entre modules acycliques)**.** *Soient M et M' deux modules, et T, T' des ensembles de sorties de M et M' respectivement. On définit que (M, T) et (M', T') sont équivalents en sortie, noté $(M, T) \equiv_O (M', T')$, si et seulement si toutes les conditions sont vérifiées pour appliquer le théorème 10 sur les modules M, M' et les sous-ensembles T, T'.*

Cette relation d'équivalence permet la construction de classes d'équivalence. Ainsi, pour M un module acyclique et T un sous-ensemble de ses automates, on définit par $C_{(M,T)} = \{(M', T') | \ (M', T') \equiv_O (M, T)\}$ la classe d'équivalence qui contient (M, T).

Parmi les très nombreuses classes d'équivalence possibles, nous notons de prime abord l'existence de classes triviales. Ces classes triviales d'équivalence ne sont composées que de fonctions de sortie de délai 1. Cette contrainte provoque une restriction drastique de la structure des modules qui les réalisent, puisqu'aucun automate ne peut influencer un automate dont la fonction de sortie est de délai 1.

Propriété 17. *Soit (M, T) tel que les fonctions de sortie de T soient toutes de délai 1. La classe $C_{(M,T)}$ ne contient que des paires (M', T') égales à (M, T) au renommage des automates et des entrées près.*

Démonstration. Soit M un module et T un ensemble de sorties de M tels que la fonction de sortie de tout s dans T a pour délai 1. Puisque le délai de ces fonctions de sortie est 1, cela implique que les automates dans T ne sont influencés par aucun autre automate dans M. Donc, par la définition d'ensemble de sorties, il n'y a aucun automate dans M à part T. Cela est vérifié pour toute paire (M', T') telle que $(M', T') \equiv_O (M, T)$. De plus, comme toutes les fonctions de sortie de M et M' sont équivalentes, toutes les fonctions locales de M et M' sont également équivalentes, au renommage des automates et entrées près. Cela s'applique pour l'entièreté de la classe $C_{(M,T)}$. □

Chacune des classes d'équivalence non triviales fait preuve d'une très grande liberté de structure parmi ses modules. En effet, pour M un module acyclique et T un ensemble de sorties de M, il existe des paires (M', T') équivalentes en sortie à (M, T) pour M' un module de taille arbitraire, et ce malgré la restriction qui fait de T un ensemble de sortie.

Propriété 18. *Soit M un module et T un ensemble de sorties. Pour tout entier N, il existe (M', T') tel que $(M, T) \equiv_O (M', T')$ et $|M'| > N$.*

Démonstration. Soit (M, T) une paire telle que M possède au moins une fonction de sortie O_s avec délai au moins 2. Cela implique qu'il existe un ensemble d'automates non vide qui influence l'automate s. Soit s' un tel automate. Nous pouvons produire une paire (M', T) équivalente en sortie à (M, T), telle que $|M| > |M'|$. Pour ce faire, on construit le module M' comme une copie du module M, auquel on rajoute un automate s'' dont la fonction locale est une copie de la fonction locale de l'automate s'. De plus, dans la fonction locale de l'automate s du module M', on substitue chaque instance de la variable $x_{s'}$ par $(x_{s'} \wedge x_{s''})$. Puisque $f_{s'} = f_{s''}$, on a $O_{s'} = O_{s''}$, ce qui implique que l'évaluation de la fonction de sortie de s dans M' reste équivalente à celle s dans M. Nous avons ainsi créé un module disposant d'un automate supplémentaire et équivalent en sortie au module initial. □

6.3 Modules 1-pour-1

Parmi les nombreuses classes d'équivalence qui organisent les modules acycliques et leurs sorties, nous nous intéressons ici aux classes produites par les modules acycliques qui ne définissent qu'une seule entrée et qu'une seule sortie, ou $|I| = |T| = 1$. Comme nous allons le voir, ces cas particuliers représentent un grand nombre de familles de réseaux d'automates.

L'ensemble des réseaux d'automates qui peuvent se décomposer simplement en un module 1-pour-1 sont facilement caractérisés comme les réseaux qui possèdent un automate en particulier qui, une fois retiré, laisse le graphe d'interaction du réseau acyclique.

Propriété 19. *Soit F un réseau d'automates. Les deux propositions suivantes sont équivalentes :*

— *Le cardinal d'un feedback vertex set minimal du graphe d'interaction de F est 1 ou moins.*
— *Il existe un module 1-pour-1 M et une fonction de branchement $\mathtt{w}$ tels que $\circlearrowleft_{\mathtt{w}} M = F$.*

Démonstration. Supposons que le graphe d'interaction de F dispose d'un feedback vertex set de taille 0. Alors F est déjà un module acyclique, et l'on peut rajouter une entrée redondante α pour définir un module 1-pour-1 M. Pour $\mathtt{w}$ la fonction de sortie définie sur α qui y associe une valeur arbitraire, on observe $\circlearrowleft_{\mathtt{w}} M = F$.

Supposons que le graphe d'interaction de F dispose d'un feedback vertex set de taille 1. Soit $s \in S$ l'automate concerné. On construit le module M défini sur l'ensemble des automates de F et sur l'ensemble d'entrées $I = \{\alpha\}$ tel que pour tout automate s', la fonction locale $f'_{s'}$ dans M est définie par $f'_{s'}(x' \cdot i') = f_s((x' \cdot i') \circ g)$, où g est la fonction $g : \{S\} \to \{S \cup I\}$ telle que

$$g(r) = \begin{cases} \alpha & \text{si } r = s \\ r & \text{sinon} \end{cases}.$$

Nous observons que, dans M, aucun automate n'est influencé par s. En effet, l'influence de la variable s a été remplacée par g par l'influence de l'entrée α. Nous en concluons que M est acyclique : en effet, par hypothèse, tout cycle dans le graphe d'interaction de M doit passer par s. Or, comme aucun sommet n'est influencé par s, aucun cycle n'existe dans le graphe d'interaction de M. Le module M dispose d'une seule entrée, α et d'une seule sortie, s. Pour conclure cette inférence, nous construisons la fonction de branchement $\mathtt{w} : \{\alpha\} \to S$ telle que $\mathtt{w}(\alpha) = s$. Le module $\circlearrowleft_{\mathtt{w}} M$ n'a pas d'entrée et dispose des automates S, et pour tout $s' \in S$, sa fonction locale $f''_{s'}$ vérifie

$$f''_{s'}(x'') = f'_{s'}(x'' \circ \hat{\mathtt{w}})$$
$$f''_{s'}(x'') = f_{s'}(x'' \circ \hat{\mathtt{w}} \circ g).$$

On observe que $\hat{\mathtt{w}} \circ g(s) = \hat{\mathtt{w}}(\alpha) = s$ et que, pour tout s' différent de s, $\hat{\mathtt{w}} \circ g(s') = \hat{\mathtt{w}}(s') = s'$, et $\hat{\mathtt{w}} \circ g$ est donc la fonction identité sur S. En conclusion, $f''_{s'}(x'') = f_{s'}(x'')$ pour tout $s' \in S$, et $\circlearrowleft_{\mathtt{w}} M = F$.

Supposons maintenant qu'il existe un module 1-pour-1 M et une fonction de branchement $\mathtt{w}$ tels que $\circlearrowleft_{\mathtt{w}} M = F$. Puisque le module M n'a qu'une seule entrée et une seule sortie, les domaine et image de $\mathtt{w}$ sont des singletons. Soit s l'automate inclus dans cette image. On peut définir une fonction g définie comme précédemment qui transforme F en M. Puisque M est acyclique, cela signifie que tout cycle dans le graphe d'interaction de F passe par s. Supposons qu'il existe un tel cycle : alors le graphe d'interaction de F dispose d'un feedback vertex set de taille 1 qui contient s. Si aucun tel cycle n'existe, alors, par définition, le graphe d'interaction de F dispose d'un feedback vertex set de taille 0. □

Ces modules ont la particularité de posséder une dynamique bien plus facile à pré-

dire, ce qui est illustré par les problèmes suivants, conçus comme des cas particuliers des problèmes développés en fin de section 6.2.3.

▸ ONE-TO-ONE MODULE ATTRACTOR PROBLEM
Entrée: Un module acyclique booléen M, une seule entrée et n automates, une fonction de branchement `w` et un nombre c encodé en unaire.
Promesse: L'encodage des fonctions locales de M ne comporte que des variables essentielles.
Question: Existe-t-il un attracteur de taille c dans la dynamique parallèle du réseau d'automates $\circlearrowleft_{\mathtt{w}} M$?

▸ ONE-TO-ONE MODULE FIXED POINT PROBLEM
Entrée: Un module acyclique booléen M, une seule entrée et n automates et une fonction de branchement `w`.
Promesse: L'encodage des fonctions locales de M ne comporte que des variables essentielles.
Question: Existe-t-il une configuration x telle que $\circlearrowleft_{\mathtt{w}} M(x) = x$?

Ces problèmes sont associés aux théorèmes suivants.

Théorème 11 (PERROT, PERROTIN et SENÉ 2020a)**.** *Le problème One-to-one Module Attractor est NP-complet.*

Démonstration. Ce problème est dans NP car, depuis toute configuration, on peut vérifier que cette configuration fait partie d'un attracteur de taille c en mettant à jour le réseau c fois, et en vérifiant que la configuration obtenue est la même que la configuration initiale.

Pour en prouver la NP-difficulté, nous présentons une réduction depuis le problème SAT. Pour f une formule à m variables nommées $v_1, \dots, v_m$, on construit un module 1-pour-1 avec $m+e+1$ automates (pour e borné par une $2m$) tel que, lorsque la sortie de ce module est branchée à son entrée, sa dynamique dispose d'un attracteur de taille $c = m+e+1$ si et seulement si f est satisfaisable.

Le module 1-pour-1, nommé M, est composé de deux parties. La première partie est un *ruban*, composé de $m+e$ automates nommés $t_1, \dots, t_{m+e}$ avec e le plus petit nombre tel que $m+e+1$ est un nombre premier (la valeur de e peut être calculée efficacement par l'algorithme décrit par (AGRAWAL, KAYAL et SAXENA 2004)). Pour tout $1 < k \leq m+e$, on définit la fonction locale $f_{t_k}(x) = t_{k-1}$, et $f_{t_1}(x) = \alpha$ avec α l'entrée unique du module. Pour $i \in \{1, \dots, m\}$, l'état de l'automate t_i encode l'évaluation de la variable v_i.

La seconde partie du réseau est composée d'un automate unique nommé q, qui est la sortie du réseau sur laquelle `w` branche l'entrée α. Cet automate a le rôle de laisser le ruban de taille $m+e$ devenir un ruban rotatif de taille $m+e+1$, ou d'arrêter le processus et de forcer le réseau à converger vers le point fixe 0^{m+e+1}. Sa fonction

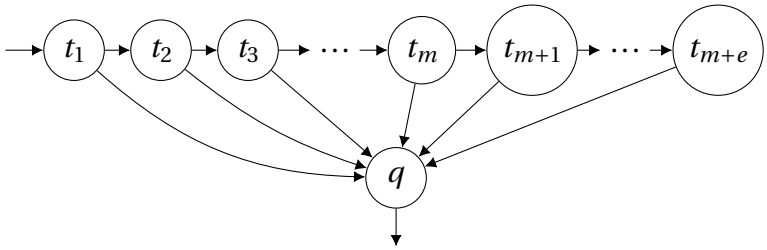

FIGURE 6.5 – Illustration du module M développé dans la preuve du théorème 11. Si f dispose d'une évaluation positive, alors l'automate q permet au ruban de taille $m+e$ de devenir un ruban rotatif de taille $m+e+1$, et sinon f_q s'évalue à 0 et toute configuration converge vers le point fixe 0^{m+e}.

locale est comme suit :

$$f_q(x) = \begin{cases} x_{t_{m+e}} & \text{si les automates } t_1,\dots,t_m \text{ encodent} \\ & \qquad \text{une évaluation positive de } f \\ & \text{ou si un décalage de cette configuration encode} \\ & \qquad \text{une évaluation positive de } f, \\ 0 & \text{sinon.} \end{cases}$$

Puisque le module M est acyclique, l'automate q ne peut qu'indirectement connaître son propre état par le branchement `w`. Après ce branchement, le sommet q possède l'information de l'entièreté du réseau, et peut donc décider si un certain décalage du ruban permet d'évaluer la formule f à vrai. Par conséquent, s'il existe une évaluation positive de f, alors il existe une configuration du réseau qui se met en rotation et en découle un attracteur de taille $\mathcal{O}(m)$. Si le réseau dispose d'un attracteur de taille supérieure à 1, alors, par la nature première de la taille du réseau, cet attracteur sera de taille $m+e+1$ et décrira une évaluation positive de f. La formule f est insatisfaisable si et seulement si toute configuration du réseau converge vers le point fixe 0^{m+e+1}.

Cette construction est illustrée par la figure 6.5, et est de taille polynomiale. □

Corollaire 4 (PERROT, PERROTIN et SENÉ 2020a)**.** *Le problème One-to-one Module Fixed Point est dans P.*

Démonstration. Ce résultat est une application du théorème 8. □

6.3.1 Les registres à décalage à rétroaction linéaire comme des modules 1-pour-1

Les registres à décalage à rétroaction linéaire sont des modèles de circuits étudiés principalement pour leur capacité à générer des séquences pseudo-aléatoires. Ces circuits disposent d'une rangée de registres qui retiennent une valeur booléenne. Lors

de la mise à jour, l'ensemble des valeurs est déplacée vers l'avant : la nouvelle valeur du premier registre est calculée par une fonction linéaire de l'état général du système.

Ces objets sont facilement modélisés par des réseaux d'automates. Nous définissons l'ensemble des réseaux d'automates qui les modélisent comme suit.

Définition 72 (Réseau d'automates à décalage à rétroaction)**.** *Un réseau d'automates est un* réseau d'automates à décalage à rétroaction *si et seulement si son réseau d'interaction dispose d'un automate s dont la fonction locale est arbitraire, et dont le reste des automates peut être décrit comme une séquence $(s_1, s_2, \ldots, s_{n-1})$ qui vérifie que $f_{s_1}(x) = x_s$, et $f_{s_i}(x) = x_{s_{i-1}}$ pour tout i tel que $1 < i < n$.*

Nous ne nommons pas ces objets "réseaux d'automates à décalage à rétroaction *linéaire*" car ce "*linéaire*" signifie, dans le cas des registres du même nom, que la fonction de réatroaction est une fonction linéaire. Les résultats qui suivent ne font sens que si l'on se permet de considérer toute fonction de rétroaction, et cela justifie notre appellation.

La caractérisation de ces réseaux par le biais de modules 1-pour-1 est triviale, et repose sur le remplacement de l'influence de l'automate s par une l'influence d'une entrée.

Propriété 20. *Soit F un réseau d'automates à décalage à rétroaction. Il existe M un module ayclique 1-pour-1 et une fonction de branchement $\mathtt{w}$ telle que $F = \circlearrowleft_{\mathtt{w}} M$.*

Démonstration. On observe que le réseau d'interaction de F dispose toujours d'un feedback vertex set défini par $\{s\}$, car tout cycle de ce graphe passe trivialement par s. La propriété est obtenue par application de la propriété 19. □

Si cette propriété permet d'observer que les réseaux d'automates à décalage à rétroaction sont bien caractérisés par les modules 1-pour-1, nous pouvons également démontrer que la fonction de sortie de tout module 1-pour-1 peut être réalisée par un réseau d'automates à décalage à rétroaction. Nous décrivons ce fait par la propriété suivante.

Propriété 21. *Soit O une fonction de sortie définie sur une unique entrée. Il existe un module 1-pour-1 M et une fonction de branchement $\mathtt{w}$ tels que $\circlearrowleft_{\mathtt{w}} M$ est un réseau d'automates à décalage à rétroaction et la fonction de sortie de M est égale à O.*

Démonstration. Soit O une fonction de sortie définie sur l'entrée α, et d le délai de O. On peut facilement composer M un module qui réalise O de la façon suivante. Le module M est défini sur les automates $S = \{s, s_1, s_2, \ldots, s_{d-1}\}$ et l'entrée $I = \{\alpha\}$ tels que $(s_1, s_2, \ldots, s_{d-1})$ est une séquence d'automates dont les fonctions locales vérifient $f_{s_1}(x) = i_\alpha$, et $f_{s_i}(x) = x_{s_{i-1}}$ pour tout i tel que $1 < i < d$. La fonction locale de l'automate s est définie comme la fonction O dans laquelle toute instance de la variable α_1 est substituée par la variable i_α, et pour tout k tel que $1 < k \leq d$, la variable α_d est substituée par la variable $x_{s_{k-1}}$. Soit $\mathtt{w} : \{\alpha\} \to S$ la fonction qui vérifie $\mathtt{w}(\alpha) = s$.

On observe que $\circlearrowleft_{\mathtt{w}} M$ est un réseau d'automate à décalage à rétroaction linéaire. De plus, la fonction de sortie de l'automate s dans M réalise par construction la fonction de sortie O : cela se voit facilement par le fait que la fonction de sortie de l'automate x_k se trouve être $O_{x_k}(J) = \alpha_k$. □

Cette propriété exprime pour nous une forme d'universalité intrinsèque des réseaux d'automates à décalage à rétroaction dans la famille plus large des réseaux d'automates obtenus par le branchement récursif d'un module 1-pour-1. En effet, pour tout réseau obtenu par le branchement récursif d'un module 1-pour-1, il existe par l'application de la propriété 20 un réseau d'automates à décalage à rétroaction qui définit une fonction de sortie équivalente. Par application du théorème 10, ce réseau d'automates à décalage à rétroaction disposera d'attracteurs isomorphes. Cette notion d'universalité présuppose que le calcul pertinent d'un réseau d'automates est l'ensemble de ses attracteurs.

6.3.2 Les fleurs d'automates comme des modules 1-pour-1

Parmi les autres familles incluses dans cette classe, nous pouvons nommer les ⊕-BAF, ou fleurs d'automates booléens localement non-monotones telles que définies dans (ALCOLEI, PERROT et SENÉ 2016). Nous reproduisons ici une définition plus générale que la définition initiale donnée dans (DIDIER et REMY 2012) :

Définition 73 (Fleur d'automates)**.** *Un réseau d'automate est une* fleur *si et seulement si son graphe d'interaction est connexe, et si tous ses sommets sauf un ont une arité entrante et sortante de un.*

Les fleurs ne sont pas des modules acycliques par cette définition. Il existe cependant une transformation intuitive de ces réseaux en modules acycliques 1-pour-1.

Propriété 22. *Soit F une fleur d'automates. Il existe M un module ayclique 1-pour-1 et une fonction de branchement* $\mathtt{w}$ *telle que $F = \circlearrowleft_{\mathtt{w}} M$.*

Démonstration. Soit F une fleur d'automates et soit s le sommet dont l'arité sortante et entrante n'est pas un. On observe que l'ensemble $\{s\}$ est un feedback vertex set du graphe d'interaction de F. Pour le voir, supposons qu'il existe un cycle dans le graphe d'interaction de F qui ne contient pas s. Par définition, tous les sommets de ce cycle ont pour arité entrante et sortante un. Cela signifie que ce cycle est fermé du reste du réseau, et donc que ce cycle et le sommet s sont sur des sous-graphes non connectés, ce qui est en contradiction avec l'hypothèse que le graphe d'interaction de F est connexe.

Le graphe d'interaction d'une fleur d'automates définit toujours un feedback vertex set de taille 1. Nous déduisons le résultat par application de la propriété 19. □

On remarque que les fleurs d'automates sont une généralisation des réseaux d'automates à décalage à rétroaction. Cela permet d'obtenir trivialement que les fleurs

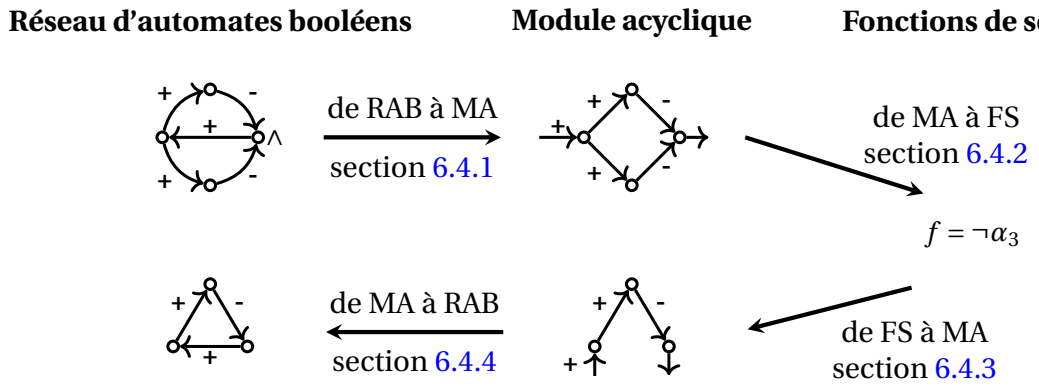

FIGURE 6.6 – Illustration de la méthode d'optimisation de module acyclique, qui permet l'optimisation de réseaux d'automates booléens pour le calcul de leur dynamique. Chaque étape de cette méthode est représentée par une flèche, qui est annotée de la section qui la décrit.

d'automates sont aussi intrinsèquement universelles que les réseaux d'automates à décalage à rétroaction.

Propriété 23. *Soit O une fonction de sortie définie sur une unique entrée. Il existe un module 1-pour-1 M et une fonction de branchement $\mathtt{w}$ tels que $\circlearrowleft_{\mathtt{w}} M$ est une fleur d'automates et la fonction de sortie de M est égale à O.*

Démonstration. Puisque tout réseau d'automates à décalage à rétroaction est une fleur d'automates, on obtient le résultat simplement par application de la propriété 21. □

6.4 Optimisation d'un module acyclique

La définition de classes d'équivalence nous pousse à l'observation d'une propriété naturelle. Pour chaque classe, il existe un ensemble de modules de cette classe qui ont une taille minimale. Cette observation nous inspire un processus d'optimisation de réseaux d'automates, qui repose sur une méthode d'optimisation de modules acycliques. Pour F un réseau d'automates, cette méthode procède par les étapes suivantes, qui sont également représentées en figure 6.6 :

1. Construire un module acyclique M tel que $\circlearrowleft_{\mathtt{w}} M = F$, pour $\mathtt{w}$ une fonction de branchement.
2. Extraire T l'ensemble de sorties qui permet la recomposition de F.
3. Trouver (M', T') tel que $(M, T) \equiv_O (M', T')$ et M' minimal en taille.
4. Recomposer M' par un branchement équivalent à $\mathtt{w}$, au renommage des automates et des entrées près.

Il ressort de cette méthode et pour tout réseau d'automate F un nouveau réseau F', de taille possiblement plus petite, et qui, par définition de l'équivalence en sortie, possède des attracteurs isomorphes aux attracteurs de F. Ainsi, cette méthode est motivée par le calcul des attracteurs d'un réseau d'automate, qui est une procédure au coût exponentiel en la taille du réseau. Nous proposons d'explorer les détails de cette méthode qui permet de réduire dans certains cas la taille des réseaux étudiés tout en préservant leurs attracteurs en dynamique parallèle.

Pour se faire, nous présentons chaque étape de cette méthode par le biais d'un problème fonctionnel, accompagné par notre analyse de la complexité de ce problème, lorsqu'applicable. Ces problèmes de complexité sont, pour la plupart, des problèmes fonctionnels avec promesse, dont la définition est détaillée en section 1.8.6. Cette promesse consiste à exprimer que les fonctions locales des modules et réseaux d'automates passés en entrée de ces modèles sont encodés de façon à n'avoir que des variables essentielles. Cela veut dire que l'ensemble des automates et entrées qui influencent la mise à jour d'un automate donné sont exactement les automates et entrées dont les variables sont présentes dans l'encodage de la fonction locale de cet automate. Cette hypothèse est forte : en effet, sa vérification est un procédé coNP-difficile. L'utilité de cette promesse est que le graphe d'interaction du réseau considéré peut alors être dessiné en temps polynomial.

Nous justifions l'utilisation de cette promesse en deux points : premièrement, la procédure d'optimisation que nous développons prend pour contexte les applications des réseaux d'automates booléens dans un contexte biologique ; dans ces applications, les réseaux étudiés sont créés de façon qui ne permet pas l'apparition de variables redondantes. Secondement, l'emploi de cette promesse nous permet l'observation de problèmes difficiles, et l'exhibition de cette difficulté sans avoir à considérer le cas particulier des variables redondantes nous semble pertinent pour comprendre les limites de notre méthode.

Les exemples suivants accompagnent la définition de notre méthode et illustrent chaque étape qui la compose.

Exemple 50. *Pour $S_A = \{a, b, c, d\}$, soit F_A le réseau d'automates booléens défini par les fonctions locales $f_a(x) = x_d$, $f_b(x) = f_c(x) = x_a$, et $f_d(x) = \neg x_b \vee \neg x_c$. Le graphe d'interaction de F_A est représenté en figure 6.7.*

Exemple 51. *Pour $S_B = \{St, Sl, Sk, Pp, Ru, S9, C, C25, M, C^*\}$, soit F_B le réseau d'automates booléens défini par les fonctions locales $f_{St}(x) = \neg x_{St}$, $f_{Sl}(x) = \neg x_{Sl} \vee x_{C^*}$, $f_{Sk}(x) = x_{St} \vee \neg x_{Sk}$, $f_{Pp}(x) = x_{Sl} \vee \neg x_{Pp}$, $f_{Ru}(x) = f_{S9}(x) = \neg x_{Sk} \vee x_{Pp} \vee \neg x_C \vee \neg x_{C^*}$, $f_C(x) = \neg x_{Ru} \vee \neg x_{S9} \vee \neg x_{Sl}$, $f_{C25}(x) = \neg x_{Pp} \vee x_C$, $f_M(x) = x_{Pp} \vee \neg x_C$, and $f_{C^*}(x) = \neg x_{Ru} \vee \neg x_{S9} \vee x_{C25} \vee \neg x_M$. Le graphe d'interaction de F_B est représenté en figure 6.8.*

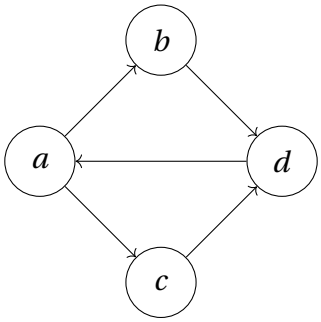

FIGURE 6.7 – Graphe d'interaction du réseau d'automates F_A défini dans l'exemple 50.

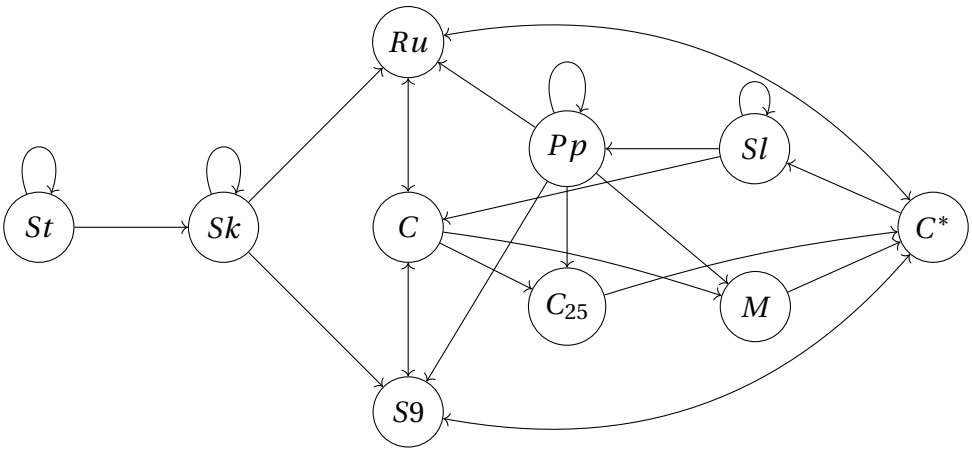

FIGURE 6.8 – Graphe d'interaction du réseau d'automates F_B défini dans l'exemple 51.

6.4.1 Transformation d'un réseau d'automate en module acyclique

La première étape de notre processus consiste à déplier, ou découper, transformer un réseau d'automates F en un module acyclique. Nous formalisons cette étape par le biais d'un problème fonctionnel.

▸ FUNCTIONAL ACYCLIC UNFOLDING PROBLEM

Entrée: Un réseau d'automates F et un entier k.
Promesse: L'encodage des fonctions locales de F ne comporte que des variables essentielles.
Sortie: Un module acyclique M avec au plus k entrées et une fonction de branchement récursif $\mathtt{w}$ tels que $\circlearrowleft_{\mathtt{w}} M = F$.

Théorème 12 (PERROT, PERROTIN et SENÉ 2020b)**.** *Le problème Functional Acyclic Unfolding est dans FNP.*

Démonstration. Rappellons que la promesse de ce problème permet le calcul du

graphe d'interaction de F en temps polynomial.

Considérons l'algorithme non déterministe suivant : en première étape, deviner M et $\mathtt{w}$; vérifier ensuite que le nombre d'entrées de M n'est pas supérieur à k, et que le module $\circlearrowleft_{\mathtt{w}} M$ est syntaxiquement égal à F.

Cet algorithme est calculable en temps polynomial non déterministe car un branchement récursif n'opère qu'une simple substitution de variables. Peu importe la façon dont les fonctions locales de F sont encodées en entrée, on pourra choisir M et $\mathtt{w}$ tels que les fonctions locales de $\circlearrowleft_{\mathtt{w}} M$ sont encodées de façon identique à F. □

Théorème 13 (PERROT, PERROTIN et SENÉ 2020b)**.** *Le problème Functional Acyclic Unfolding est NP-difficile.*

Démonstration. Nous procédons à une réduction many-one polynomiale depuis le problème Feedback Vertex Set. Soit f une fonction qui, pour tout graphe G et tout nombre k tel que (G,k) est une instance du problème Feeback Vertex Set, donne une instance (F,k) du problème Functionnal Acyclic Unfolding telle que $S = V(G)$, et f_s est une fonction OR de tous les automates s' dans S tels que (s',s') est une arête de G. Cette construction implique que le graphe d'interaction et le graphe G soient identiques, et est clairement calculable en temps polynomial. On en déduit que l'instance (G,k) est positive si et seulement si (F,k) possède une solution.

Cette réduction fonctionne sur l'observation que remplacer l'influence d'un automate par une entrée dans un module retire du graphe d'interaction tous les cycles qui passent par cet automate. En effet, une fois que l'influence de l'automate est remplacée par une entrée sur l'ensemble du réseau, cet automate n'a plus aucune arête sortante dans le graphe d'interaction. En termes de l'existence de cycles dans le graphe d'interaction, ce remplacement d'influence est équivalent à retirer l'automate du module. □

Cette transformation est toujours possible grâce à l'application de la propriété 14. Afin d'assurer que le module obtenu n'aura qu'un nombre minimal de sorties, nous procédons ici par réduction vers le problème Feedback Vertex Set.

En sortie de ce problème, nous obtenons M et $\mathtt{w}$. L'ensemble de sorties T considéré est simplement l'ensemble des sommets dont l'influence a été remplacée par des entrées, ce qui correspond à l'ensemble $\mathrm{codom}(\mathtt{w})$.

Exemple 52. *Nous considérons S_A et F_A de l'exemple 50. Pour $I_A = \{\alpha\}$, soit M_A le module acyclique défini par les fonctions locales $f'_a(x) = x_\alpha$, $f'_b(x) = f'_c(x) = x_a$, et $f'_d(x) = \neg x_b \vee \neg x_c$. Le module M_A est une réponse valide à l'instance $F_A, k = 1$ du problème Functional Acyclic Unfolding, et il n'existe pas de solution pour $k = 0$. Le graphe d'interaction du module M_A est représenté en figure 6.9.*

Exemple 53. *Nous considérons S_B et F_B de l'exemple 51. Pour $I_B = \{\alpha_{St}, \alpha_{Sl}, \alpha_{Sk}, \alpha_{Pp}, \alpha_C, \alpha_{C^*}\}$, soit M_B le module acyclique défini par les fonctions locales $f'_{St}(x) = \neg x_{\alpha_{St}}$, $f'_{Sl}(x) = \neg x_{\alpha_{Sl}} \vee x_{\alpha_{C^*}}$, $f'_{Sk}(x) = x_{\alpha_{St}} \vee \neg x_{\alpha_{Sk}}$, $f'_{Pp}(x) = x_{\alpha_{Sl}} \vee \neg x_{\alpha_{Pp}}$, $f'_{Ru}(x) = f_{S9}(x) = \neg x_{\alpha_{Sk}} \vee x_{\alpha_{Pp}} \vee \neg x_{\alpha_C} \vee \neg x_{\alpha_{C^*}}$, $f'_C(x) = \neg x_{Ru} \vee \neg x_{S9} \vee \neg x_{\alpha_{Sl}}$, $f'_{C25}(x) = \neg x_{\alpha_{Pp}} \vee x_{\alpha_C}$,*

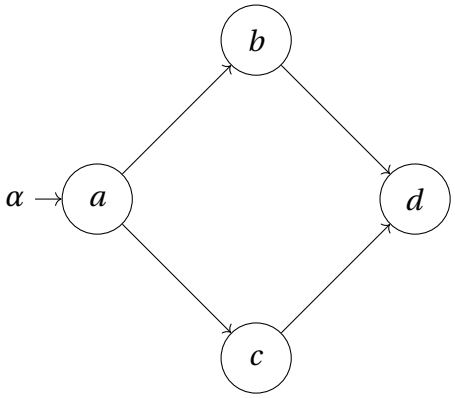

FIGURE 6.9 – Graphe d'interaction du module M_A défini dans l'exemple 52.

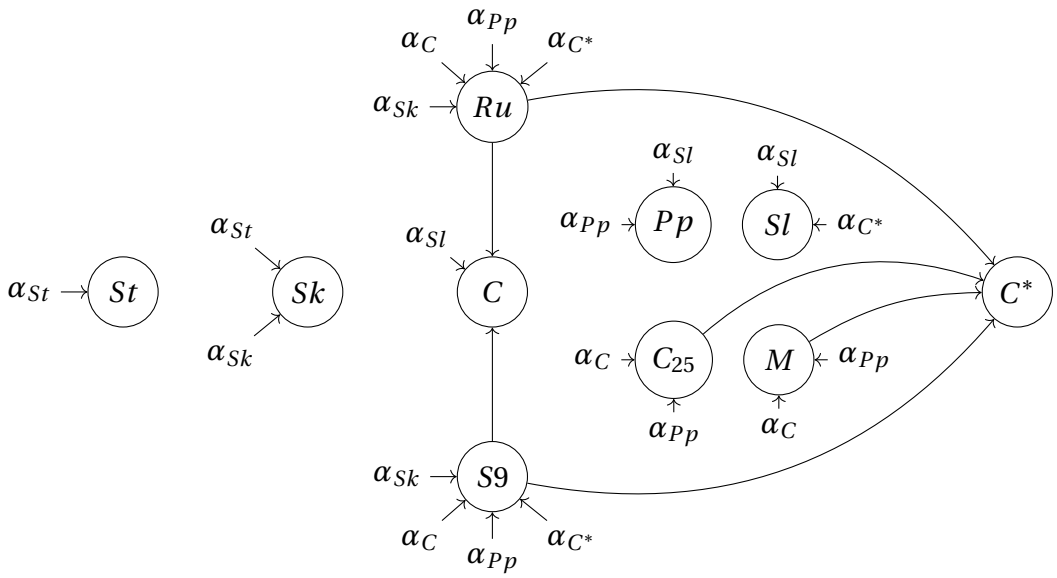

FIGURE 6.10 – Graphe d'interaction du module M_B défini dans l'exemple 53.

$f'_M(x) = x_{\alpha_{Pp}} \vee \neg x_{\alpha_C}$, *et* $f'_{C^*}(x) = \neg x_{Ru} \vee \neg x_{S9} \vee x_{C25} \vee \neg x_M$. *Le module* M_B *est une réponse valide à l'instance* $F_B, k = 6$ *du problème Functional Acyclic Unfolding, et il n'existe pas de solution pour* $k < 6$. *Le graphe d'interaction du module* M_B *est représenté en figure 6.10.*

6.4.2 Calcul efficace des fonctions de sortie

Soit (M, T) la paire d'éléments obtenus en sortie du problème précédent. Afin d'obtenir une paire (M', T') possiblement de plus petite taille, nous procédons en deux étapes. La première consiste à calculer les fonctions de sortie des automates

T du module M sous la forme de circuits booléens. La seconde consiste à générer un module minimal M' qui calcule les même fonctions de sortie sur un ensemble d'automates T'.

Comme discuté en section 6.2.2, nous encodons les fonctions de sortie par des circuits booléens car cela nous permet de les calculer en temps polynomial, là où les encoder par des formules booléennes ne nous donne pas cette assurance.

La première partie de la minimisation du module est le calcul de ses fonctions de sortie, que nous caractérisons par le problème fonctionnel suivant.

▸ OUTPUT CIRCUIT COMPUTATION PROBLEM

Entrée: Un module acyclique M avec $X \subseteq S$ un sous-ensemble de ses automates.

Promesse: L'encodage des fonctions locales de M ne comporte que des variables essentielles.

Sortie: Une fonction de sortie pour chaque automate de X, encodée sous la forme d'un circuit booléen.

Ainsi formulée, cette partie du processus d'optimisation peut être calculée en temps polynomial.

Théorème 14 (PERROT, PERROTIN et SENÉ 2020b)**.** *Le problème Output Circuit Computation est dans FP.*

Démonstration. La promesse de ce problème permet la construction du graphe d'interaction de M en temps polynomial. Le résultat est obtenu par application de la propriété 16. □

Exemple 54. *Nous considérons le module M_A de l'exemple 52. Soit $M_A, X_A = \{d\}$ une instance du problème Output Circuit Computation. Le circuit $O_d = \neg\alpha_3$ est une réponse valide à cette instance.*

Exemple 55. *Nous considérons le module M_B de l'exemple 53. Soit $M_B, X_B = \{St, Sk, Sl, Pp, C, C^*\}$ une instance du problème Output Circuit Computation. L'ensemble des circuits $O_{St} = \neg\alpha_{St,1}$, $O_{Sl} = \neg\alpha_{Sl,1} \vee \alpha_{C^*,1}$, $O_{Sk} = \alpha_{St,1} \vee \neg\alpha_{Sk,1}$, $O_{Pp} = \alpha_{Sl,1} \vee \neg\alpha_{Pp,1}$, $O_C = (\alpha_{Sk,2} \wedge \neg\alpha_{Pp,2} \wedge \alpha_{C,2} \wedge \alpha_{C^*,2}) \vee \neg\alpha_{Sl,1}$ et $O_{C^*} = \alpha_{C,2} \vee \neg\alpha_{Pp,2}$ est une solution valide à cette instance.*

6.4.3 Génération d'un module acyclique optimal

Une fois que nous obtenons une collection de fonctions de sortie pour le sous-ensemble T du module M, nous pouvons nous intéresser à contruire un module acyclique optimal qui réalise ces fonctions de sortie. Cette question est caractérisée par le problème suivant.

▸ MODULE SYNTHESIS PROBLEM
Entrée: Un ensemble I d'entrées, un ensemble de fonctions de sortie encodées par des circuits booléens O définies sur les entrées I et k un entier.
Sortie: Un module acyclique M avec au plus k automates tel que chaque fonction de O est une fonction de sortie d'au moins un automate de M.

Notre compréhension de ce problème ne nous permet pour l'instant que de dessiner des bornes de complexité assez lâches. Il est en effet très facile de prouver les résultats suivants.

Théorème 15 (PERROT, PERROTIN et SENÉ 2020b)**.** *Le problème Module Synthesis est coNP-difficile.*

Démonstration. On considère f une formule instance du problème Tautology, avec I l'ensemble de variables propositionnelles contenues dans f. Soit f' la fonction de sortie obtenue à partir de f en remplaçant chaque instance d'une variable α par l'équivalent de délai 1, α_1. Nous définissons également $\mathbf{1}$ la fonction de sortie constante à 1. Nous composons $O = \{f', \mathbf{1}\}$ et $k = 1$. Cette instance a une solution si et seulement si ces deux fonctions de sortie peuvent être réalisées par un seul automate, autrement si et seulement si f' est une tautologie. Cette instance peut clairement être construite en temps polynomial, et toute solution positive de cette instance implique que f est une tautologie. □

Théorème 16 (PERROT, PERROTIN et SENÉ 2020b)**.** *Le problème Module Synthesis est dans* FNP^{coNP}.

Démonstration. On considère l'algorithme suivant. Premièrement, deviner un module acyclique M de taille k. Calculer ensuite chaque fonction de sortie du module en temps polynomial. Vérifier ensuite que chaque fonction dans O est équivalente à au moins une de ces fonctions de sortie, ce qui peut être fait avec au plus $|M| \times |O|$ appels à un oracle coNP. □

L'avancée actuelle de notre travail ne nous permet pas de raffiner cette borne de complexité plus précisément. Bien que les deux bornes ici formalisées se trouvent être des résultats relativement simples, les tentatives de preuve de raffinement de ces bornes que nous avons explorées n'ont retourné aucun résultat.

Question ouverte 2. *Quelle est la complexité précise du problème Module Synthesis ?*

Prouver que ce problème est dans FcoNP nécessite une réalisation de ce problème par un algorithme de même complexité. Un tel algorithme dispose des objets I, O et k en entrée, et a la capacité de construire de façon non déterministe un objet de taille polynomiale en la taille de ces entrées, puis pourra vérifier en temps polynomial déterministe que cet objet implique que l'instance n'a pas de solution valide. Cependant, résoudre le problème Module Synthesis nécessite au moins la génération d'un module acyclique M, à partir duquel il est un problème coNP de vérifier que ce module réalise

les bonne fonctions de sortie car, en effet, il s'agit de la complexité du problème de l'égalité de deux circuits booléens. Il nous semble raisonnable de conclure que tout calcul d'une solution en temps FcoNP semble impossible.

Une autre possibilité de preuve est un algorithme qui procède en générant une solution naïve, et en optimisant cette solution par la fusion de ses automates deux à deux. Cette approche nécessite cependant la résolution d'un problème de la forme suivante.

▶ MODULE LOCAL FUSION PROBLEM

Entrée: Une module acyclique M et a, b deux automates différents de M.

Question: Existe-t-il une fonction locale f_c telle qu'il existe un module acyclique M' défini sur les automates $S \cup \{c\} \setminus \{a, b\}$, tel que $f_c \equiv f'_c$ et $O_s \equiv O'_s$ pour tout s dans $S \setminus \{a, b\}$?

La résolution d'un ensemble d'instances du problème Module Local Fusion est une étape nécessaire pour tout algorithme qui optimise un module par la fusion de ses automates. Ce problème est au moins un problème dans $\mathrm{NP}^{\mathrm{coNP}}$. En effet, son énoncé repose sur le problème de deviner un nouveau module M', tel que ce module vérifie l'égalité des fonctions f_c et f'_c, ainsi que l'égalité des fonctions de sortie, ce qui, en soi, est un ensemble de problèmes coNP-difficiles. Un algorithme reposant sur la fusion ne pourra donc pas être d'une complexité inférieure à $\mathrm{NP}^{\mathrm{coNP}}$.

Prouver que ce problème est $\mathrm{NP}^{\mathrm{coNP}}$-difficile, aussi appelé $\Sigma_2 P$-difficile, requiert la réduction d'un problème $\Sigma_2 P$-complet vers le problème Module Synthesis. Un exemple classique de problème $\Sigma_2 P$-complet est le problème de la satisfaisabilité d'une formule $\exists x_1, x_2, \ldots, \forall y_1, y_2, \ldots, \Phi(x, y)$, pour Φ une formule booléenne. Nous n'avons pas d'intuition sur la façon dont le problème Module Synthesis permettrait l'encodage d'un problème aussi complexe.

Nous concluons cette discussion sur la complexité évasive du problème Module Synthesis en le rapprochant à d'autres problèmes en surface très similaires. Un problème similaire est le problème Circuit Minimisation, qui repose sur le calcul d'un circuit booléen de taille optimale qui calcule une table de vérité donnée en entrée. Ce problème est trivialement prouvé d'être dans NP, mais la connaissance actuelle du domaine ne permet pas de décider s'il s'agit d'un problème dans P ou d'un problème NP-complet.

Bien que les problèmes Circuit Minimisation et Module Synthesis semblent similaires, ces similarités sont en vérité superficielles. En effet, si les modules acycliques et les circuits booléens ont en commun d'être représentés comme des graphes orientés acycliques, sont considérés d'avoir des entrées et des sorties et dont le calcul est caractérisé par des fonctions booléennes, ces similarités ne sont d'aucune aide pour comparer les problèmes de leur optimisation en taille. Si optimiser un circuit booléen repose sur la question de réaliser une fonction booléenne de façon la plus compacte et efficace en utilisant un ensemble limité de portes, optimiser un module acyclique demande d'optimiser un calcul qui est réalisé à travers le temps. Les contraintes qui concernent l'optimisation d'un circuit booléen reposent sur la structure même des

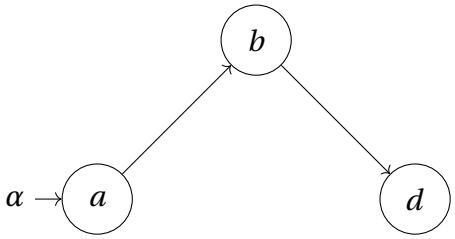

FIGURE 6.11 – Graphe d'interaction du module M'_A défini dans l'exemple 56.

fonctions booléennes, alors que les contraintes qui concernent l'optimisation d'un module acyclique reposent sur la difficulté de créer un flot d'information qui permet la réalisation d'une sortie composée d'entrées à différents délais. Ainsi, ces problèmes nous semblent trop différents pour permettre une réduction de l'un vers l'autre, et ce malgré leurs nombreuses similarités.

Le problème Module Synthesis est un problème difficile, et sa résolution dans le cas général parait compromettre la recherche d'une solution d'optimisation efficace. Cependant, son application sur de petites instances permet d'observer de réelles optimisations dans la taille de certain des réseaux étudiés. De telles optimisations sont illustrées par les exemples suivants.

Exemple 56. *Nous considérons la fonction de sortie O_d définie dans l'exemple 54. Pour $S'_A = \{a, b, d\}$ et $I_A = \{\alpha\}$, soit M'_A le module défini par les fonctions locales $f''_a = x_\alpha$, $f''_b = x_a$ et $f_d = \neg x_b$. Le module M'_A est une réponse valide à l'instance $I_A, \{O_d\}, k = 3$ du problème Module Synthesis. Le graphe d'interaction du module M'_A est représenté en figure 6.11.*

Exemple 57. *Nous considérons les fonctions de sorties $O_B = \{O_{St}, O_{Sl}, O_{Sk}, O_{Pp}, O_C, O_{C^*}\}$ définies dans l'exemple 55. Pour $S'_B = \{St, Sl, Sk, Pp, Ru, C25, C^*\}$ et $I_B = \{\alpha_{St}, \alpha_{Sl}, \alpha_{Sk}, \alpha_{Pp}, \alpha_C, \alpha_{C^*}\}$, soit M'_B le module défini par les fonctions locales $f''_{St}(x) = \neg x_{\alpha_{St}}$, $f''_{Sl}(x) = \neg x_{\alpha_{Sl}} \vee x_{\alpha_{C^*}}$, $f''_{Sk}(x) = x_{\alpha_{St}} \vee \neg x_{\alpha_{Sk}}$, $f''_{Pp}(x) = x_{\alpha_{Sl}} \vee \neg x_{\alpha_{Pp}}$, $f''_{Ru}(x) = \neg x_{\alpha_{Sk}} \vee x_{\alpha_{Pp}} \vee \neg x_{\alpha_C} \vee \neg x_{\alpha_{C^*}}$, $f''_C(x) = \neg x_{Ru} \vee \neg x_{\alpha_{Sl}}$, $f''_{C25}(x) = \neg x_{\alpha_{Pp}} \vee x_{\alpha_C}$, et $f'_{C^*}(x) = x_{C25}$. Le module M'_B est une réponse valide à l'instance $I_B, O_B, k = 8$ du problème Module Synthesis. Le graphe d'interaction du module M'_B est représenté en figure 6.12.*

6.4.4 Recomposition du réseau final

Enfin, la dernière étape de l'optimisation d'un réseau est la recomposition d'un réseau d'automates à partir du module optimal généré à la section précédente.

Cette recomposition est opérée simplement par l'application d'un branchement récursif qui est défini par l'étape précédente du processus qui a transformée un réseau d'automates en un module acyclique. Lors de cette opération, l'influence d'un ensemble d'automates a été remplacé par des entrées. L'étape finale du processus

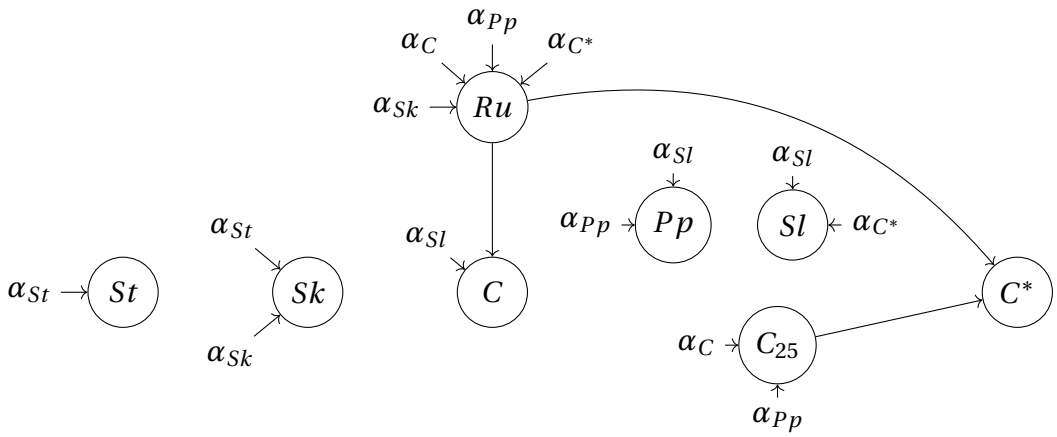

FIGURE 6.12 – Graphe d'interaction du module M'_B défini dans l'exemple 57.

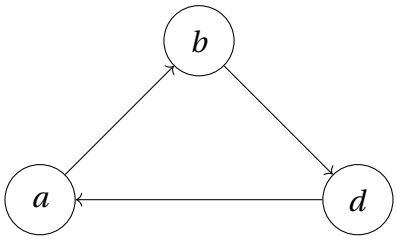

FIGURE 6.13 – Graphe d'interaction du réseau d'automates S'_A défini dans l'exemple 58.

consiste en un branchement de ces entrées vers les automates dont l'influence a été substituée.

Exemple 58. *Nous considérons le module acyclique M'_A de l'exemple 56. Soit $\mathtt{w}_A$ la fonction de branchement qui vérifie $\mathtt{w}_A(\alpha) = d$. Le réseau d'automates booléens $\circlearrowleft_{\mathtt{w}_A} M'_A$ est défini sur les automates $S'_A = \{a, b, d\}$ et les fonctions locales $f'''_a(x) = x_d$, $f'''_b(x) = x_a$, et $f'''_d(x) = \neg x_b$. Le graphe d'interaction de ce module est représenté en figure 6.13.*

Exemple 59. *Nous considérons le module acyclique M'_B de l'exemple 57. Soit $\mathtt{w}_B$ la fonction de branchement qui vérifie $\mathtt{w}_B(\alpha_s) = s$ pour tout s dans X_B. Le réseau d'automates booléens $\circlearrowleft_{\mathtt{w}_B} M'_B$ est défini sur les automates $S'_B = \{St, Sl, Sk, Pp, Ru, C25, C^*\}$ et le fonctions locales $f'''_{St}(x) = \neg x_{St}$, $f'''_{Sl}(x) = \neg x_{Sl} \vee x_{C^*}$, $f'''_{Sk}(x) = x_{St} \vee \neg x_{Sk}$, $f'''_{Pp}(x) = x_{Sl} \vee \neg x_{Pp}$, $f'''_{Ru}(x) = \neg x_{Sk} \vee x_{Pp} \vee \neg x_C \vee \neg x_{C^*}$, $f'''_C(x) = \neg x_{Ru} \vee \neg x_{Sl}$, $f'''_{C25}(x) = \neg x_{Pp} \vee x_C$, et $f'_{C^*}(x) = x_{C25}$. Le graphe d'interaction de ce module est représenté en figure 6.14.*

À la fin de tout ce processus, nous obtenons un réseau d'automates dont la taille est égale ou inférieure à la taille du réseau initial. De plus, nous avons la garantie par application du théorème 10 que les attracteurs du réseau final sont isomorphes

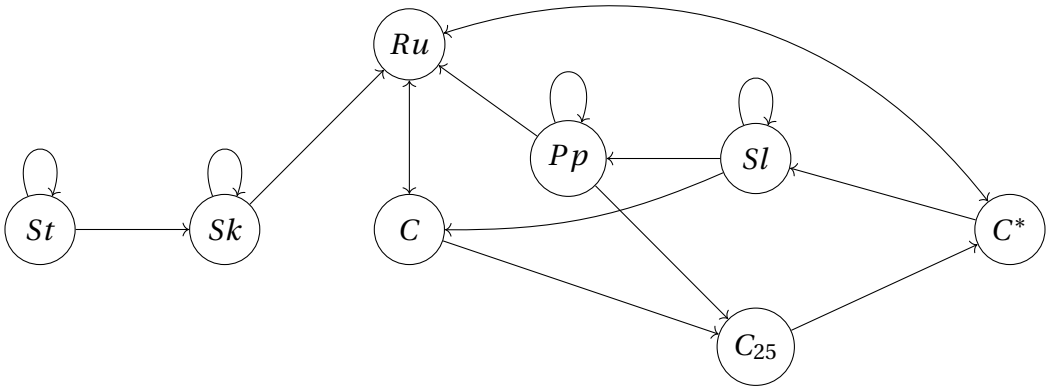

FIGURE 6.14 – Graphe d'interaction du réseau d'automates S'_B défini dans l'exemple 59.

aux attracteurs du réseau initial. L'application de ce théorème est faite sur la paire de modules acycliques constituée du module obtenu après application du problème Functional Acyclic Unfolding sur le réseau initial et le module obtenu par application du problème du Module Synthesis sur le premier module. Cette paire de modules acycliques possèdent des fonctions de sortie équivalentes au renommage des entrées et des automates près, et sont sont reconstitués par un branchement récursif équivalents par le même renommage des entrées et des automates en la paire de réseaux constituée du réseau initial, et du réseau final.

Dans la pratique, nous n'avons pas de garantie que ce processus permette une véritable optimisation. Cependant, nous illustrons certains cas positifs par le biais des séries d'exemples 50, 52, 54, 56, 58 et 51, 53, 55, 57, 59. La première série illustre l'application de notre méthode sur un réseau de taille 4 simple, construit pour posséder deux automates redondant qui sont "compactés" par notre méthode, qui obtient un réseau final de taille 3 décrit dans l'exemple 58. La seconde série illustre l'application de notre méthode sur un réseau plus complexe, emprunté dans (DAVIDICH et BORNHOLDT 2008), qui modélise le comportement de la division d'une cellule de levure. Ce réseau, initialement composé de 10 automates (exemple 51), est optimisé pour n'en contenir plus que 8 (exemple 59). Notre première série d'exemples permet donc le calcul des attracteurs du réseau initial par le développement d'un graphe de la dynamique de taille $2^3 = 8$ plutôt que par le développement d'un graphe de la dynamique de taille $2^4 = 16$. Similairement, la seconde série permet le calcul des attracteurs du réseau initial par le développement d'un graphe de la dynamique de taille $2^8 = 256$ plutôt que par le développement d'un graphe de la dynamique de taille $2^10 = 1024$.

En plus de permettre une réduction par une puissance de 2 du graphe de la dynamique parallèle du réseau étudié, cette méthode met en avant une notion formelle de la redondance dans un réseau d'automates. En effet, il semble que si cette méthode permet l'optimisation d'un réseau, c'est qu'il existe un ensemble d'automates dans ce

réseau dont le rôle peut être effectué par un ensemble d'automates plus petit. Bien que la méthode présentée ne s'applique que pour le mode de mise à jour parallèle, elle souligne que le nombre de telles "redondances" dans un réseau d'automates booléen implique une borne supérieure sur la taille totale des attracteurs de sa dynamique, car chaque redondance implique l'existence d'un réseau plus petit dont les attracteurs sont équivalents.

6.4.5 Cas des modules 1-pour-1

Bien que la complexité du cas général de l'optimisation d'un réseau d'automates reste une question difficile, nous détaillons dans cette section la complexité précise de l'optimisation d'un réseau qui peut être déplié en modules 1-pour-1. La définition de ces réseaux, qui sont caractérisés par un feedback vertex set minimal de taille 1 ou moins, est disponible en section 6.3.

Lors de l'application de la première étape de la procédure d'optimisation développée en section 6.4, l'utilisation d'un tel réseau permet l'obtension d'un module 1-pour-1, dont la seconde étape de la procédure pourra extraire une fonction de sortie définie sur une seule entrée.

Cette unique fonction de sortie peut être transformée en un module acyclique optimal par l'application de la propriété 21. En effet, cette propriété énonce qu'il existe un module 1-pour-1 qui réalise cette fonction, tel que sa taille est le délai de cette fonction. La construction de ce module peut être opérée en temps polynomial en la taille de l'encodage de la fonction de sortie, et nous montrons que sa taille est optimale.

▸ 1-FOR-1 MODULE SYNTHESIS PROBLEM
Entrée: Une fonction de sortie O définie sur une entrée unique α, et de délai d.
Sortie: Un module acyclique M optimal tel que la fonction O est la fonction de sortie d'au moins un automate dans M.

Théorème 17 (PERROT, PERROTIN et SENÉ 2020b)**.** *Le problème 1-for-1 Module Synthesis est dans FP.*

Démonstration. Pour $\mathtt{d}$ le délai de la fonction de sortie O, nous observons qu'il n'est pas possible de produire un module de taille inférieure à $\mathtt{d}$: la taille de la plus longue chaîne dans le module donne une borne supérieure à la variable de plus grand délai parmi les fonctions de sortie du module.

Nous reproduisons la construction détaillée dans la preuve de la propriété 21. Cette construction est de taille $\mathtt{d}$, et ne requiert que l'assemblage d'une chaîne d'automates de taille $\mathtt{d}-1$ avec des fonctions locales de taille constante et un automate dont la fonction locale est obtenue par substitution depuis O. Ce procédé est polynomial et optimal. □

Ce résultat est pertinent lorsque l'on considère que les modules 1-pour-1 ne sont pas une réduction triviale des modules acycliques, mais généralisent de nombreuses

classes de réseaux dont la caractérisation des attracteurs nous échappe encore. Ce résultat permet de réduire l'étude des attracteurs de cette famille de réseaux à l'étude des attracteurs des réseaux d'automates à décalage à rétroaction, qui sont optimaux en taille.

6.5 Conclusion

Dans ce chapitre, nous avons étudié le cas particulier que représentent les modules acycliques. Ces objets qui généralisent les réseaux d'automates acycliques – dont le comportement est simple – possèdent un comportement suffisamment complexe pour que leur étude soit pertinente dans le cadre de la compréhension des réseaux d'automates, mais également suffisamment simple pour permettre l'élaboration de résultats.

S'il existe une méthode d'optimisation efficace des modules 1-pour-1 mais que le cas général est difficile à caractériser, la meilleure direction future semble l'étude progressive et attentive des modules k-pour-k. Pour chaque entier k, notre formalisme offre une famille de modules plus générale que la famille précédente, qui englobe une portion plus large de la faune des réseaux d'automates booléens, et dont la difficulté de l'optimisation est plus grande. Cette étude offre une hiérarchie intuitive, permettant de représenter l'évolution de la difficulté de ces questions mais aussi de nombreuses autres – la nature et les propriétés de leur fonctions de sortie, le nombre total de cycles que les réseaux associés admettent – à travers l'augmentation de la connectivité des modules étudiés.

Perspectives

Entre nos questionnements sur la généralisation de la simulation et sur les problèmes pratiques des modules et leur restriction acyclique, ce manuscrit ouvre un éventail de problématiques. Nous proposons, pour conclure, d'énoncer quelques lignes de perspective sur les questions qui semblent les plus prometteuses pour l'avenir.

Généralisation de la simulation Notre proposition de définition générale, inspirée de la bisimulation, fait partie d'un travail de collaboration plus large, dont le projet est d'explorer une définition formelle de simulation et de simulation intrinsèque dans les modèles de calcul naturel. Le travail qui attend le développement de cette définition est l'application systématique et disciplinée de cette définition "candidate" aux très nombreux domaines dans lesquels elle puise son intuition : au domaine des automates cellulaires et leur définition très générale de simulation (Delorme, Mazoyer, Ollinger et al. 2011a ; Delorme, Mazoyer, Ollinger et al. 2011b ; Ollinger 2012), au domaine des réseaux d'automates (Melliti, Regnault, Richard et al. 2016 ; Bridoux, Castillo-Ramirez et Gadouleau 2020 ; Perrot, Perrotin et Sené 2018, aux modèles auto-assemblants (Doty, Lutz, Patitz et al. 2012 ; Meunier, Patitz, Summers et al. 2014) et beaucoup d'autres. Profondément liées aux questions d'universalité et d'universalité intrinsèque, il est attendu que cette généralisation se prouve difficilement et qu'il n'existe pas de définition unique qui respecte toutes les intuitions du domaine. Si tel est le cas, un travail d'approfondissement de ces intuitions est à attendre qui permette une distinction plus subtile entre définitions de simulation lorsque cela est possible. Ce travail d'envergure tient la clé d'une cartographie intime des domaines du calcul naturel, touchant par des définitions aussi générales que possible les intuitions profondes qui régissent les notions de calcul intrinsèque.

L'extension du rapprochement entre simulation et complexité Si notre approche ne s'est intéressée qu'aux classes P, NP et coNP, de très nombreuses questions se posent quant au prolongement de cette approche. Toutes les notions de réductions font-elles sens dans ce cadre plus général ? Est-il possible de définir la traduction de classes telles que NC, FP, ou $\#\mathsf{P}$? Peut-on prouver que $\mathsf{P}_{\preccurlyeq} \neq \mathsf{NP}_{\preccurlyeq}$? Quels sont les critères qui permettent à un système d'être complet pour une classe de simulation ? Dans tous les cas, ces questions ne peuvent pas être sereinement explorées si la définition sous-jacente de simulation n'est pas satisfaisante.

Simulation, calcul et émergence Dans les notions évoquées dans notre discussion sur les intuitions de la simulation, il a été fait référence à des notions de philosophie, pour lesquelles une littérature existe. Dans le futur, il peut être pertinent de rechercher plus en profondeur si les questions abordées font lien avec des notions déjà développées. L'émergence, par exemple, est une notion au centre de nombreux débats entre monistes, réductionnistes ou dualistes. Entre émergence faible et émergence forte, la définition même du concept n'admet pas de convention claire. Pouvons-nous apporter une contribution plus approfondie à ces discussions à l'aide de la théorie du calcul et de la simulation ?

Modules k-pour-k Les modules acycliques s'avèrent être un outil intéressant qui permettent d'aborder les questions de la complexité de la dynamique des réseaux d'automates d'une façon novatrice. Nous sommes intéressés d'approfondir les modules 1-pour-1 en une extension nommée modules k-pour-k, avec k un entier. Cette perspective propose naturellement de se reposer chaque question posée pour les modules 1-pour-1, c'est-à-dire la difficulté de la prédiction de leur dynamique ainsi que de leur optimisation, au cas plus large des modules k-pour-k. Tout comme les familles des fleurs d'automates et des registres à décalage à rétroaction peuvent être catégorisés comme se décomposant sous la forme de modules 1-pour-1, nous envisageons d'établir un système de catégorisation plus large, divisant la faune des familles de réseaux d'automates notables par leur capacité à être décomposées sous la forme de modules k-pour-k. Cette hiérarchie est projetée de pouvoir permettre la généralisation de problèmes d'optimisation de ces familles afin de calculer leur dynamique.

Modules acycliques et modes de mise à jour Notre caractérisation du calcul des modules acyclique se limite à leur étude sous le mode de mise à jour parallèle. À l'avenir, il sera intéressant de généraliser les notions de fonctions de sortie pour n'importe quel mode de mise à jour, ce qui modifiera les fonctions de sortie obtenues. Cela amène à vérifier si le théorème 10 peut être généralisé en dehors du mode de mise à jour parallèle. Si une version de ce théorème se trouve applicable entre des modules acycliques au mode de mise à jour différent, on pourra obtenir un outil permettant de caractériser sous quelles mises à jour quels réseaux disposent d'une certaine dynamique limite.

Redondance d'automates et bornes sur les attracteurs Notre méthode d'optimisation permet de réduire la taille d'un réseau, tout en conservant des attracteurs équivalent. Cette méthode se repose sur la présence de redondances parmi les automates du réseau étudié ; ces redoncances impliquent qu'il existe un réseau plus petit capable du même calcul limite, et cela implique une borne sur la taille totale des attracteurs du réseau initial. Est-il possible de caractériser de façon plus précise cette notion de redondance ? Et, autre question importante, si la redondance implique une borne sur la taille totale des attracteurs, peut-on prouver qu'il existe une redondance dans un réseau dès que la taille totale de ses attracteurs est en dessous de cette borne ?

Bibliographie

[AKS04] M. AGRAWAL, N. KAYAL et N. SAXENA. « PRIMES is in P ». In : *Annals of mathematics* 160.2 (2004), p. 781-793 (cf. p. 129).

[APS16] A. ALCOLEI, K. PERROT et S. SENÉ. « On the flora of asynchronous locally non-monotonic Boolean automata networks ». In : *Proceedings of SASB'15*. T. 326. ENTCS. 2016, p. 3-25 (cf. p. 63, 132).

[AS19] H. ALKHUDHAYR et J. STEGGLES. « A compositional framework for Boolean networks ». In : *Biosystems* 186 (2019) (cf. p. 79).

[Alo85] N. ALON. « Asynchronous threshold networks ». In : *Graphs and Combinatorics* 1 (1985), p. 305-310 (cf. p. 63).

[Alo03] U. ALON. « Biological networks : the tinkerer as an engineer ». In : *Science* 301 (2003), p. 1866-1867 (cf. p. 79).

[Ara08] J. ARACENA. « Maximum number of fixed points in regulatory Boolean networks ». In : *Bulletin of mathematical biology* 70 (2008), p. 1398-1409 (cf. p. 63, 76).

[Ara+11] J. ARACENA, E. FANCHON, M. MONTALVA et al. « Combinatorics on update digraphs in Boolean networks ». In : *Discrete Applied Mathematics* 159.6 (2011), p. 401-409 (cf. p. 76).

[Ara+09] J. ARACENA, E. GOLES, A. MOREIRA et al. « On the robustness of update schedules in Boolean networks ». In : *Biosystems* 97.1 (2009), p. 1-8 (cf. p. 76).

[AGS13] J. ARACENA, L. GÓMEZ et L. SALINAS. « Limit cycles and update digraphs in Boolean networks ». In : *Discrete Applied Mathematics* 161 (2013), p. 1-12 (cf. p. 69, 76).

[ARS17] J. ARACENA, A. RICHARD et L. SALINAS. « Number of fixed points and disjoint cycles in monotone Boolean networks ». In : *SIAM Journal on Discrete Mathematics* 31 (2017), p. 1702-1725 (cf. p. 63, 67, 76).

[AFW07] P. ARRIGHI, R. FARGETTON et Z. WANG. « Intrinsically universal one-dimensional quantum cellular automata in two flavours ». In : *CoRR* abs/0704.3961 (2007) (cf. p. 10).

[BP17] J. C. BAEZ et B. S. POLLARD. « A compositional framework for reaction networks ». In : *Reviews in Mathematical Physics* 29.09 (2017) (cf. p. 79).

[BTW87] P. BAK, C. TANG et K. WIESENFELD. « Self-organized criticality : An explanation of the 1/f noise ». In : *Physical review letters* 59.4 (1987), p. 381 (cf. p. 10).

[Bec+18] F. BECKER, D. MALDONADO, N. OLLINGER et al. « Universality in Freezing Cellular Automata ». In : *Conference on Computability in Europe*. Springer. 2018, p. 50-59 (cf. p. 31).

[Ber66] R. BERGER. *The undecidability of the domino problem*. 66. American Mathematical Society, 1966 (cf. p. 10).

[Ber+03] G. BERNOT, J. GUESPIN-MICHEL, J.-P. COMET et al. « Modeling, observability and experiment, a case study ». In : *Proceedings of the Dieppe Spring school on Modelling and simulation of biological processes in the context of genomics*. 2003, p. 49-55 (cf. p. 10).

[BT09] G. BERNOT et F. TAHI. « Behaviour preservation of a biological regulatory network when embedded into a larger network ». In : *Fundamenta Informaticae* 91 (2009), p. 463-485 (cf. p. 79).

[Boa14] P. V. E. BOAS. « Machine models and simulations ». In : *Handbook of Theoretical Computer Science, volume A* (2014), p. 1-66 (cf. p. 29).

[Bos12] F. BOSCHETTI. « Causality, emergence, computation and unreasonable expectations ». In : *Synthese* 185.2 (2012), p. 187-194 (cf. p. 60).

[Bri19] F. BRIDOUX. « Intrinsic simulations and complexities in automata networks ». Thèse de doct. Aix-Marseile Université, 2019 (cf. p. 33).

[BCG20] F. BRIDOUX, A. CASTILLO-RAMIREZ et M. GADOULEAU. « Complete simulation of automata networks ». In : *Journal of Computer and System Sciences* 109 (2020), p. 1-21 (cf. p. 33, 147).

[Bri+19] F. BRIDOUX, A. DURBEC, K. PERROT et al. « Complexity of maximum fixed point problem in Boolean Networks ». In : *Conference on Computability in Europe*. T. 11558. LNCS. Springer, 2019, p. 132-143 (cf. p. 63).

[BGT20] F. BRIDOUX, M. GADOULEAU et G. THEYSSIER. « On Simulation in Automata Networks ». In : *Beyond the Horizon of Computability* (2020) (cf. p. 33).

[Bri+] F. BRIDOUX, C. GAZE-MAILLOT, K. PERROT et al. « Complexity of limit-cycle problems in Boolean networks ». In : Accepted at SOFSEM'21 (cf. p. 63).

[Bri+17] F. BRIDOUX, P. GUILLON, K. PERROT et al. In : (2017), p. 112-128 (cf. p. 33).

[Cod14] E. F CODD. *Cellular automata*. Academic Press, 2014 (cf. p. 9).

[Con+96] ES CONNOLLY, CJ. WINFREE, TA. SPRINGER et al. « Cerebral protection in homozygous null ICAM-1 mice after middle cerebral artery occlusion. Role of neutrophil adhesion in the pathogenesis of stroke. » In : *The Journal of clinical investigation* 97.1 (1996), p. 209-216 (cf. p. 10).

[Coo04] M. COOK. « Universality in elementary cellular automata ». In : *Complex Systems* 15 (2004), p. 1-40 (cf. p. 10, 31, 41, 56).

[Cru94] J. P CRUTCHFIELD. « The calculi of emergence : computation, dynamics and induction ». In : *Physica D : Nonlinear Phenomena* 75.1-3 (1994), p. 11-54 (cf. p. 60).

[DB08] M.I. DAVIDICH et S. BORNHOLDT. « Boolean network model predicts cell cycle sequence of fission yeast ». In : *PLoS One* 3 (2008), e1672 (cf. p. 63, 143).

[Del+12] F. DELAPLACE, H. KLAUDEL, T. MELLITI et al. « Analysis of modular organisation of interaction networks based on asymptotic dynamics ». In : *Proceedings of CMSB'12*. T. 7605. LNCS. 2012, p. 148-165 (cf. p. 79).

[Del+11a] M. DELORME, J. MAZOYER, N. OLLINGER et al. « Bulking I : an abstract theory of bulking ». In : *Theoretical Computer Science* 412.30 (2011), p. 3866-3880 (cf. p. 31, 147).

[Del+11b] M. DELORME, J. MAZOYER, N. OLLINGER et al. « Bulking II : Classifications of cellular automata ». In : *Theoretical Computer Science* 412.30 (2011), p. 3881-3905 (cf. p. 31, 147).

[Dem+10] J. DEMONGEOT, E. GOLES, M. MORVAN et al. « Attraction basins as gauges of robustness against boundary conditions in biological complex systems ». In : *PLoS One* 5 (2010), e11793 (cf. p. 63).

[DNS12] J. DEMONGEOT, M. NOUAL et S. SENÉ. « Combinatorics of Boolean automata circuits dynamics ». In : *Discrete Applied Mathematics* 160.4-5 (2012), p. 398-415 (cf. p. 63, 69, 76).

[DR12] G. DIDIER et E. REMY. « Relations between gene regulatory networks and cell dynamics in Boolean models ». In : *Discrete Applied Mathematics* 160.15 (2012), p. 2147-2157 (cf. p. 132).

[Dot+12] D. DOTY, JH. LUTZ, MJ. PATITZ et al. « The tile assembly model is intrinsically universal ». In : *IEEE 53rd Annual Symposium on Foundations of Computer Science* (Hyatt Regency, 20-23 oct. 2012). New Brunswick, New Jersey, USA : IEEE Computer Society, 2012, p. 302-310 (cf. p. 147).

[DFS99] R. G. DOWNEY, M. R. FELLOWS et U. STEGE. « Parameterized complexity : A framework for systematically confronting computational intractability ». In : *Contemporary trends in discrete mathematics : From DIMACS and DIMATIA to the future*. T. 49. 1999, p. 49-99 (cf. p. 25).

[Els59] B. ELSPAS. « The theory of autonomous linear sequential networks ». In : *IRE Transactions on Circuit Theory* 6.1 (1959), p. 45-60 (cf. p. 63).

[FO89] P. FLOREEN et P. ORPONEN. « Counting stable states and sizes of attraction domains in Hopfield nets is hard ». In : *Proceedings of the International Joint Conference on Neural Networks*. 1989, p. 395-399 (cf. p. 63).

[GR16] M. GADOULEAU et A. RICHARD. « Simple dynamics on graphs ». In : *Theoretical Computer Science* 628 (2016), p. 62-77 (cf. p. 67).

[GMG02] A. GAJARDO, A. MOREIRA et E. GOLES. « Complexity of Langton's ant ». In : *Discrete Applied Mathematics* 117.1-3 (2002), p. 41-50 (cf. p. 10).

[Gar70] M. GARDNER. « Mathematical games : The fantastic combinations of John Conway's new solitaire game "life" ». In : *Scientific American* 223.4 (1970), p. 120-123 (cf. p. 10).

[Gar77] M. GARDNER. « Mathematical games ». In : *Scientific American* 236.5 (1977), p. 128-138 (cf. p. 10).

[Gol06] O. GOLDREICH. « On promise problems : A survey ». In : *Theoretical computer science*. Springer, 2006, p. 254-290 (cf. p. 27).

[GS08] E. GOLES et L. SALINAS. « Comparison between parallel and serial dynamics of Boolean networks ». In : *Theoretical Computer Science* 396 (2008), p. 247-253 (cf. p. 69, 76).

[Gol+67] S. W. GOLOMB et al. *Shift register sequences*. Aegean Park Press, 1967 (cf. p. 63).

[HS08] J. R. HINDLEY et J. P. SELDIN. *Lambda-calculus and Combinators, an Introduction*. T. 13. Cambridge University Press Cambridge, 2008 (cf. p. 41).

[ILO20] R. ILANGO, B. LOFF et I. C. OLIVEIRA. « NP-hardness of circuit minimization for multi-output functions ». In : *Electronic Colloquium on Computational Complexity*. 2020, TR20-021 (cf. p. 63).

[Ily02] Y. ILYASHENKO. « Centennial history of Hilbert's 16th problem ». In : *Bulletin of the American Mathematical Society* 39.3 (2002), p. 301-354 (cf. p. 76).

[Kar72] R. M. KARP. « Reducibility among combinatorial problems ». In : *Complexity of computer computations*. Springer, 1972, p. 85-103 (cf. p. 110).

[Kau69] S. A. KAUFFMAN. « Metabolic stability and epigenesis in randomly constructed genetic nets ». In : *Journal of Theoretical Biology* 22 (1969), p. 437-467 (cf. p. 10, 63).

[KP54] S. C. KLEENE et E. L. POST. « The upper semi-lattice of degrees of recursive unsolvability ». In : *Annals of mathematics* (1954), p. 379-407 (cf. p. 63).

[Lan84] C. G LANGTON. « Self-reproduction in cellular automata ». In : *Physica D : Nonlinear Phenomena* 10.1-2 (1984), p. 135-144 (cf. p. 10).

[Lan86] C. G LANGTON. « Studying artificial life with cellular automata ». In : *Physica D : Nonlinear Phenomena* 22.1-3 (1986), p. 120-149 (cf. p. 10).

[Len+93] CS. LENT, PD. TOUGAW, W. POROD et al. « Quantum cellular automata ». In : *Nanotechnology* 4.1 (1993), p. 49 (cf. p. 10).

[Lew75] G.H. LEWES. *Problems of Life and Mind*. Trübner & Company, 1875 (cf. p. 60).

[Mar15] S. MARTIEL. « Approches informatique et mathématique des dynamiques causales de graphes ». Thèse de doct. Nice, 2015 (cf. p. 31).

[MP43] W. S MCCULLOCH et W. PITTS. « A logical calculus of the ideas immanent in nervous activity ». In : *Bulletin of Mathematical Biophysics* 5.4 (1943), p. 115-133 (cf. p. 10, 63).

[Mel+16] T. MELLITI, D. REGNAULT, A. RICHARD et al. « Asynchronous Simulation of Boolean Networks by Monotone Boolean Networks ». In : t. 9863. Lecture Notes in Computer Science. Springer, 2016, p. 182-191 (cf. p. 147).

[MA98] L. MENDOZA et E. R. ALVAREZ-BUYLLA. « Dynamics of the genetic regulatory network for *Arabidopsis thaliana* flower morphogenesis ». In : *Journal of theoretical biology* 193 (1998), p. 307-319 (cf. p. 63).

[Meu+14] P-E. MEUNIER, MJ. PATITZ, SM. SUMMERS et al. « Intrinsic universality in tile self-assembly requires cooperation ». In : *Proceedings of the 25th Annual ACM Symposium on Discrete Algorithms* (Hilton Portland and Executive Tower, 5-7 jan. 2014). Portland, Oregon, USA : ACM, 2014, p. 752-771 (cf. p. 147).

[Mil71] R. MILNER. *An algebraic definition of simulation between programs*. Citeseer, 1971 (cf. p. 41).

[Mil+02] R. MILO, S. SHEN-ORR, S. ITZKOVITZ et al. « Network motifs : simple building blocks of complex networks ». In : *Science* 298 (2002), p. 824-827 (cf. p. 79).

[NB+66] J. NEUMANN, A. W BURKS et al. *Theory of self-reproducing automata*. T. 1102024. University of Illinois press Urbana, 1966 (cf. p. 9).

[Nou12] M. NOUAL. « Updating Automata Networks ». Thèse de doct. École Normale Supérieure de Lyon, 2012 (cf. p. 63, 75).

[NS18] M. NOUAL et S. SENÉ. « Synchronism versus asynchronism in monotonic Boolean automata networks ». In : *Natural Computation* 17 (2018), p. 393-402 (cf. p. 69, 76).

[Noû+20] C. NOÛS, K. PERROT, S. SENÉ et al. « #P-completeness of counting update digraphs, cacti, and a series-parallel decomposition method ». In : *Lecture Notes in Computer Science*. Springer, 2020 (cf. p. 63, 76).

[OCo94] T. O'CONNOR. « Emergent properties ». In : *American Philosophical Quarterly* 31.2 (1994), p. 91-104 (cf. p. 60).

[Oll02] N. OLLINGER. « Automates cellulaires : structures ». Thèse de doct. École Normale Supérieure de Lyon, 2002 (cf. p. 31).

[Oll12] N. OLLINGER. « Universalities in Cellular Automata ». In : *Handbook of Natural Computing*. Sous la dir. de G. ROZENBERG, T. BÄCK et JN. KOK. Springer, 2012. Chap. 6, p. 189-229 (cf. p. 31, 147).

[ON07] N. OROS et C. L NEHANIV. « Sexyloop : Self-reproduction, evolution and sex in cellular automata ». In : *Proceedings of ALIFE'07*. IEEE. 2007, p. 130-138 (cf. p. 10).

[Orp92] P. ORPONEN. « Neural networks and complexity theory ». In : *Proceedings of MFCS'92*. T. 629. LNCS. 1992, p. 50-61 (cf. p. 63).

[PR12] L. PAULEVÉ et A. RICHARD. « Static analysis of boolean networks based on interaction graphs : A survey ». In : *Electronic Notes in Theoretical Computer Science* 284 (2012), p. 93-104 (cf. p. 67).

[PPS18] K. PERROT, P. PERROTIN et S. SENÉ. « A framework for (de)composing with Boolean automata networks ». In : *Proceedings of MCU'18*. T. 10881. LNCS. 2018, p. 121-136 (cf. p. 32, 33, 147).

[PPS20a] K. PERROT, P. PERROTIN et S. SENÉ. « On the complexity of acyclic modules in automata networks ». In : *Proceedings of TAMC'20*. Accepted, arXiv :1910.07299. 2020 (cf. p. 118-121, 129, 130).

[PPS20b] K. PERROT, P. PERROTIN et S. SENÉ. « Optimising attractor computation in Boolean automata networks ». In : *arXiv :2005.14531* (2020). Submitted (cf. p. 116, 135, 136, 138, 139, 144).

[Rem+03] E. REMY, B. MOSSÉ, C. CHAOUIYA et al. « A description of dynamical graphs associated to elementary regulatory circuits ». In : *Bioinformatics* 19.suppl_2 (2003), p. ii172-ii178 (cf. p. 75).

[Ren16] P. RENDELL. « Turing machine in Conway game of life ». In : *Designing Beauty : The Art of Cellular Automata*. Springer, 2016, p. 149-154 (cf. p. 10).

[Ric09] A. RICHARD. « Positive circuits and maximal number of fixed points in discrete dynamical systems ». In : *Discrete Applied Mathematics* 157.15 (2009), p. 3281-3288 (cf. p. 76).

[Ric10] A. RICHARD. « Negative circuits and sustained oscillations in asynchronous automata networks ». In : *Advances in Applied Mathematics* 44.4 (2010), p. 378-392 (cf. p. 75).

[Ric18] A. RICHARD. « Fixed points and connections between positive and negative cycles in Boolean networks ». In : *Discrete Applied Mathematics* 243 (2018), p. 1-10 (cf. p. 67).

[RC07] A. RICHARD et J.-P. COMET. « Necessary conditions for multistationarity in discrete dynamical systems ». In : *Discrete Applied Mathematics* 155.18 (2007), p. 2403-2413 (cf. p. 75).

[Rob80] F. ROBERT. « Iterations sur des ensembles finis et automates cellulaires contractants ». In : *Linear Algebra and its applications* 29 (1980), p. 393-412 (cf. p. 74, 107).

[Rob86] F. ROBERT. *Discrete Iterations : A Metric Study*. Springer, 1986 (cf. p. 69, 75).

[Sal14] A. SALOMAA. *Theory of automata*. Elsevier, 2014 (cf. p. 41).

[San11] D. SANGIORGI. *Introduction to bisimulation and coinduction*. Cambridge University Press, 2011 (cf. p. 41).

[Sen12] S. SENÉ. *Sur la bio-informatique des réseaux d'automates*. Habilitation à diriger des recherches. 2012 (cf. p. 75).

[Sie09] H. SIEBERT. « Dynamical and structural modularity of discrete regulatory networks ». In : *Proceedings of COMPMOD'09*. T. 6. EPTCS. 2009, p. 109-124 (cf. p. 79).

[The05] G. THEYSSIER. « Automates cellulaires : un modèle de complexités ». Thèse de doct. École Normale Supérieure de Lyon, 2005 (cf. p. 31).

[Tho73] R. THOMAS. « Boolean formalization of genetic control circuits ». In : *Journal of Theoretical Biology* 42 (1973), p. 563-585 (cf. p. 10, 63).

[Tho81] R. THOMAS. « On the relation between the logical structure of systems and their ability to generate multiple steady states or sustained oscillations ». In : *Numerical methods in the study of critical phenomena*. Springer, 1981, p. 180-193 (cf. p. 75).

[Wai74] R. T WAINWRIGHT. « Life is universal ! » In : *Proceedings of the 7th conference on Winter simulation*. 1974, p. 449-459 (cf. p. 10).

[Wan61] H. WANG. « Proving theorems by pattern recognition—II ». In : *Bell system technical journal* 40.1 (1961), p. 1-41 (cf. p. 10).

[WS18] W. J WILDMAN et F L. SHULTS. « Emergence : What Does It Mean and How Is It Relevant to Computer Engineering ? » In : *Emergent Behavior in Complex Systems Engineering : A Modeling and Simulation Approach* (2018), p. 21-34 (cf. p. 60).

[Wol84] S. WOLFRAM. « Universality and complexity in cellular automata ». In : *Physica D : Nonlinear Phenomena* 10.1 (1984), p. 1-35 (cf. p. 22).

Index