Nathalie Aubrun
Eric Duchêne
Jacques Duparc
Claude-Pierre Jeannerod
Alexandre Miquel
Aline Parreau
Nathalie Revol
Guillaume Theyssier

Informatique Mathématique
Une photographie en 2017

Bruno Salvy (éd.)

CNRS Éditions

CNRS Éditions, 2017, ISBN : 978-2-271-11523-2

Dépôt légal : janvier 2017

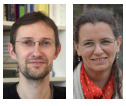
Sommaire

Les auteurs

Nathalie Aubrun est chargée de recherche CNRS au Laboratoire de l'Informatique de Parallélisme (ENS de Lyon) et Guillaume Theyssier est chargé de recherche CNRS à l'Institut de Mathématiques de Marseille (Aix Marseille Université). Leur chapitre « Pavages et automates cellulaires » présente un panorama des résultats d'indécidabilité sur les pavages par tuiles de Wang et automates cellulaires.

Eric Duchêne et Aline Parreau sont respectivement maître de conférences et chargée de recherche CNRS au laboratoire LIRIS de l'Université Lyon 1. Leur chapitre « Une introduction aux jeux combinatoires » donne les bases de la théorie de Conway sur les jeux à deux joueurs et information parfaite. Les notions les plus importantes, illustrées d'exemples concrets, seront présentées. La question de la complexité algorithmique de résolution d'un jeu sera également abordée, en lien avec les problématiques phares du domaine.

Jacques Duparc est Professeur à l'Université de Lausanne. Son chapitre « Jeux, topologie et automates » présente un aperçu de quelques méthodes développées en topologie et théorie des jeux. Il vous initiera à la complexité topologique des langages ω-réguliers.



Claude-Pierre Jeannerod et Nathalie Revol sont chargés de recherche Inria au laboratoire LIP de l'École normale supérieure de Lyon, Université de Lyon. Leur chapitre « Analyser et encadrer les erreurs dues à l'arithmétique flottante » explique pourquoi les calculs en arithmétique flottante sont entachés d'erreurs d'arrondi. Il présente ensuite deux types d'approche pour étudier et majorer ces erreurs.

Alexandre Miquel est professeur à l'Institut de Mathématiques et de Statistiques Rafael Laguardia, à la Faculté d'Ingénierie de l'Université de la République (Montevideo, Uruguay). Son chapitre « Une introduction à la réalisabilité classique » présente les bases de la correspondance preuves/programmes en logique classique. Il vous montrera comment extraire un programme à partir d'une preuve mathématique… même non constructive !

Préface

Ce livre est le cinquième de la série « Informatique Mathématique : une photographie en… ». Depuis 2013, les supports des cours donnés à l'École des Jeunes Chercheurs/Chercheuses en Informatique Mathématique sont édités. Ces livres constituent autant de photographies instantanées de notre domaine, qui illustrent sa diversité, sa vitalité et sa perpétuelle évolution. Les Écoles des Jeunes Chercheurs/Chercheuses en Informatique Mathématique sont organisées chaque année depuis la création en 2006 du GDR de même nom, le GDR IM. L'édition 2017 de École, à laquelle correspond ce livre, est organisée à Lyon du 23 au 27 janvier.

Le but de cette école annuelle est de donner une formation complémentaire de haut niveau à de jeunes chercheurs (qui sont en général à plus ou moins deux ans de leur soutenance de thèse) : cela peut être pour eux une mise à niveau dans certains domaines, ou une véritable ouverture vers de nouvelles problématiques. Ceci est important : qui travaillera exactement sur son sujet de thèse dans 10 ans ? En leur montrant l'état de la recherche dans des domaines voisins de leur spécialité, l'école permet d'élargir la culture scientifique des doctorants et leur donne des outils qui leur permettront de mieux s'adapter à des environnements variés. Elle contribue ainsi à faciliter leur recrutement et leur mobilité. En leur donnant l'occasion de se rencontrer et de présenter leurs travaux, elle contribue aussi à créer une communauté de jeunes scientifiques autour des thèmes de l'IM.

Nous remercions les auteurs d'avoir accepté avec enthousiasme de s'atteler à ce travail d'écriture et pour l'ouvrage de grande qualité qui en résulte. Nous tenons aussi à remercier chaleureusement tous les membres du comité d'organisation de l'École Jeunes Chercheurs 2017.

À Paris, le 10 novembre 2016,

Arnaud Durand et Jean-Michel Muller, Co-directeurs du GDR IM.
Bruno Salvy, Coordinateur de l'ouvrage.

Remerciements

Les auteurs et le comité scientifique remercient très chaleureusement les relecteurs pour leur travail et leurs remarques sur les versions préliminaires de cet ouvrage : Sebastian Barbieri, Valérie Berthé, Simon Castellan, Arnaud Durand, Olivier Finkel, Charles Grellois, Philippe Langlois, Hugues de Lassus Saint-Geniès, Olivier Laurent, William Lochet, Antoine Plet, et Nicolas Trotignon.

Les éditeurs remercient Valérie Berthé pour son aide à la mise en forme de cet ouvrage.

Chapitre 1

Pavages et automates cellulaires

Nathalie Aubrun
Guillaume Theyssier

Les décalages de type fini sont des ensembles de coloriages d'une structure discrète qui vérifient certaines contraintes locales. Leur étude est fortement liée à celle des automates cellulaires, un modèle de système dynamique discret dont la règle d'évolution est elle aussi locale. Dans ce cours nous présentons divers résultats sur ces objets, notamment des résultats de calculabilité. Nous montrons comment le passage de la dimension 1 *aux dimensions supérieures modifie radicalement les propriétés combinatoires et dynamiques de ces deux objets.*

1.1 Introduction

La dynamique symbolique est l'étude des sous-décalages, *i.e.* des ensembles de coloriages de $\mathbb{Z}$ ou $\mathbb{Z}^2$ par un alphabet fini $\mathcal{A}$ qui respectent des contraintes locales, données par des motifs interdits. Les sous-décalages peuvent être vus non seulement comme des modèles discrets de systèmes dynamiques, mais aussi comme un modèle de calcul. Les sous-décalages de type fini (SFT) forment une classe particulièrement intéressante, à la fois parce qu'ils peuvent être décrits par une quantité finie d'information, et parce qu'ils peuvent modéliser des phénomènes réels.

La dynamique symbolique classique, définie dans l'article fondateur [94] afin d'étudier une discrétisation de systèmes dynamiques, s'est historiquement concentrée sur le cas unidimensionnel [81] et a été ensuite

généralisée aux dimensions supérieures ou égales à 2 [80]. Mais augmenter la dimension n'est pas sans conséquence sur les propriétés combinatoires et de décidabilité des SFT. Même le plus simple des problèmes de décision sur les SFT (décider si un ensemble fini de motifs interdits définit un sous-décalage vide ou non, aussi appelé problème du Domino [134]) est décidable en dimension 1 mais indécidable dès que la dimension augmente.

Un automate cellulaire est un système dynamique qui agit sur les coloriages de $\mathbb{Z}$ ou $\mathbb{Z}^2$ par une règle de modification locale appliquée uniformément. Le modèle est né dans les années 1950 avec les travaux de J. von Neumann sur l'auto-reproduction [101] et sa collaboration avec S. Ulam qui étudiait la croissance des cristaux. La simplicité de leur définition est trompeuse et cache une grande richesse de comportements. Il n'est donc pas étonnant de retrouver ces objets dans de nombreux domaines [64, 74, 16, 17] : pour un informaticien, un automate cellulaire peut être vu comme un modèle de calcul massivement parallèle ; pour un mathématicien comme un système dynamique discret, *i.e.* comme une fonction continue agissant sur un espace compact ; pour un modélisateur (physique, biologie, écologie, etc) comme une discrétisation d'une équation aux dérivées partielles. Les automates cellulaires partagent avec les sous-décalages la localité et l'uniformité, et on peut relier l'espace ou l'espace-temps des automates cellulaires à des sous-décalages particuliers. En particulier, la dimension joue également un rôle crucial dans les problèmes de décision sur automates cellulaires et le problème du Domino est la pierre angulaire de nombreux résultats dans ce cadre.

Le point de vue adopté ici est à l'interface mathématique/informatique et centré sur les liens entre les propriétés dynamiques et combinatoires de ces objets et leur capacité à calculer.

1.2 Généralités

1.2.1 Configurations, motifs et cylindres

Soit $\mathcal{A}$ un alphabet, dont les éléments sont appelés *symboles*. On considère l'ensemble des applications $x : \mathbb{Z}^d \to \mathcal{A}$, noté $\mathcal{A}^{\mathbb{Z}^d}$, dont les éléments $x \in \mathcal{A}^{\mathbb{Z}^d}$ sont des *configurations*. L'ensemble $\mathcal{A}^{\mathbb{Z}^d}$ est donc *l'ensemble des configurations*. Soit $\mathbb{U} \subset \mathbb{Z}^d$ un ensemble fini de points de $\mathbb{Z}^d$. Un *motif* est une configuration $p \in \mathcal{A}^{\mathbb{U}}$, et l'ensemble $\mathrm{supp}(p) = \mathbb{U}$ est le *support* du motif p. Le *support élémentaire de taille n* est l'hypercube $\mathbb{U}_n = [-n; n]^d$. Un motif dont le support est $\mathbb{U}_n$ pour un certain n est un *motif élémentaire*.

Si $x \in \mathcal{A}^{\mathbb{Z}^d}$ est une configuration, on note $x_{\mathbb{U}}$ le motif fini correspon-

dant à la restriction de x à $\mathbb{U}$. En particulier, si $\mathbb{U} = \{\mathbf{i}\}$ avec $\mathbf{i} \in \mathbb{Z}^d$ est réduit à un élément, on notera simplement $x_\mathbf{i}$ le symbole en position $\mathbf{i}$. Un motif $p \in \mathcal{A}^\mathbb{U}$ *apparaît* dans configuration x s'il existe $\mathbf{i} \in \mathbb{Z}^d$ de sorte que $x_{\mathbf{i}+\mathbb{U}} = p$. Dans ce cas, on note $p \sqsubset_\mathbf{i} x$ ou $p \sqsubset x$ si on ne souhaite pas préciser à quelle position le motif apparaît. Un motif $p \in \mathcal{A}^\mathbb{U}$ est un *sous-motif* de $p' \in \mathcal{A}^{\mathbb{U}'}$, ce qu'on écrira $p \sqsubseteq p'$, si $\mathbb{U} \subset \mathbb{U}'$ et $p'_\mathbb{U} = p$.

Exemple 1.2.1. *Soit $\mathcal{A}$ l'alphabet $\{\square, \blacksquare\}$. Considérons les deux configurations $x \in \mathcal{A}^{\mathbb{Z}^2}$ et $y \in \mathcal{A}^{\mathbb{Z}^2}$ dessinées sur la Figure 1.1 (seulement une portion finie est représentée). Le motif p apparaît dans x en positions $(3,1)$ et $(-1,-3)$, mais n'apparaît pas dans la portion finie de y représentée sur la Figure 1.1.*

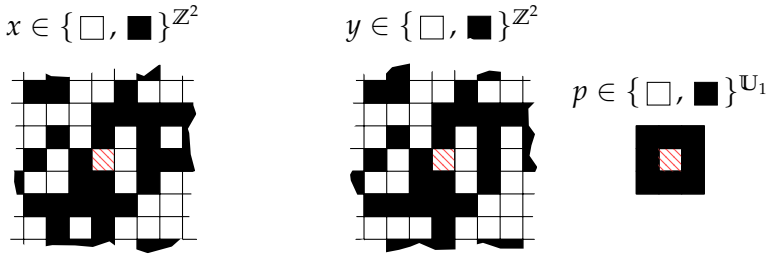


FIGURE 1.1 – *Deux configurations $x, y \in \{\square, \blacksquare\}^{\mathbb{Z}^2}$ (l'origine de $\mathbb{Z}^2$ est représentée par des hachures rouges), et un motif élémentaire $p \in \{\square, \blacksquare\}^{[-1;1]\times[-1;1]}$.*

Étant donné un motif d-dimensionnel p sur l'alphabet $\mathcal{A}$, le *cylindre* porté par p est l'ensemble de toutes les configurations qui coïncident avec p sur son support :

$$[\![p]\!] = \left\{ x \in \mathcal{A}^{\mathbb{Z}^d} \mid x_{|\operatorname{supp}(p)} = p \right\}.$$

En d'autre termes, le cylindre $[\![p]\!]$ est exactement l'ensemble des configurations infinies qui prolongent le motif p. On notera qu'un motif p peut apparaître dans une configuration x sans que celle-ci n'appartienne au cylindre $[\![p]\!]$ (comme par exemple x et p de l'Exemple 1.2.1).

1.2.2 L'espace de Cantor $\mathcal{A}^{\mathbb{Z}^d}$

L'alphabet fini $\mathcal{A}$ est naturellement muni d'une topologie, la topologie discrète, à partir de laquelle on peut construire la topologie pro-discrète sur l'ensemble des configurations $\mathcal{A}^{\mathbb{Z}^d}$ (il s'agit donc d'une topologie produit). Vu comme un espace topologique, l'ensemble des configurations $\mathcal{A}^{\mathbb{Z}^d}$ est

appelé *ensemble de Cantor*. Il existe plusieurs manières équivalentes de définir cette topologie :

séquentielle : une suite $(x^n)_{n\in\mathbb{N}}$ d'éléments de $\mathcal{A}^{\mathbb{Z}^d}$ converge vers $x \in \mathcal{A}^{\mathbb{Z}^d}$ si pour tout $\mathbf{i} \in \mathbb{Z}^d$ il existe $N_\mathbf{i} \in \mathbb{N}$ tel que $x^n_\mathbf{i} = x_i$ pour tout $n \geqslant N_\mathbf{i}$;

par base d'ouverts : l'ensemble des cylindres

$$\{[\![p]\!] \mid p \text{ est un motif sur } \mathcal{A}\}$$

est une base d'ouverts, *i.e.* les ouverts sont les unions de cylindres ;

par une métrique : $\mathcal{A}^{\mathbb{Z}^d}$ est métrisable, par exemple avec la distance d telle que $d(x,x) = 0$, et

$$d(x,y) = 2^{-\min\{|\mathbf{i}| \,|\, x_\mathbf{i} \neq y_\mathbf{i}\}} \text{ pour tous } x, y \in \mathcal{A}^{\mathbb{Z}^d},$$

avec $|\mathbf{i}| = \max_{k=1,\dots,d} |\mathbf{i}_k|$. En d'autres termes, deux configurations sont d'autant plus proches qu'elles partagent un grand motif central. Par exemple, les deux configurations de la figure 1.1 sont à distance $\frac{1}{8}$ car elles diffèrent en position $(3,0)$ mais sont égales pour toutes les positions $\mathbf{i}$ telles que $|\mathbf{i}| \leqslant 2$

Pour rappel, l'équivalence provient des liens suivants entre ouverts, suites convergentes et distance :

— une suite $(x^n)_{n\in\mathbb{N}}$ converge vers x si pour tout ouvert O contenant x il existe N tel que $n \geqslant N$ implique $x^n \in O$;
— un fermé (*i.e.* le complémentaire d'un ouvert) est un ensemble F tel que toute suite d'éléments de F qui converge, converge vers un élément de F ;
— la boule ouverte de rayon r est l'ensemble $B_r(x) = \{y \mid d(x,y) < r\}$ et ces ensembles forment une base d'ouverts ;
— une suite $(x^n)_{n\in\mathbb{N}}$ converge vers x si pour tout $\epsilon > 0$ il existe N tel que $n \geqslant N$ implique $d(x, x^n) \leqslant \epsilon$.

Notons au passage que pour tout motif $p \in \mathcal{A}^{\mathbb{U}}$ avec $\mathbb{U} \subset \mathbb{Z}^d$ fini, le cylindre $[\![p]\!]$ est un ouvert-fermé, car il peut s'écrire

$$[\![p]\!] = \mathcal{A}^{\mathbb{Z}^d} \setminus \bigcup_{q\in\mathcal{A}^{\mathbb{U}}, q\neq p} [\![q]\!].$$

Exemple 1.2.2. *Soit $\mathcal{A} = \{0,1,2\}$ et soit $\mathbf{X}$ l'ensemble des configurations de $\mathcal{A}^{\mathbb{Z}^d}$ où le symbole 2 n'apparaît pas, et $\mathbf{Y}$ sont complémentaire. Voici trois façons de voir que $\mathbf{X}$ est un fermé et $\mathbf{Y}$ un ouvert :*

— $\mathbf{Y}$ *est l'union des cylindres* $[\![p_{\mathbf{j}}]\!]$ *où* $p_{\mathbf{j}}$ *est le motif de domaine* $\{\mathbf{j}\}$ *tel que* $p_{\mathbf{j}}(\mathbf{j}) = 2$ (exercice)*;*
— *si une suite* (x^n) *d'éléments de* $\mathbf{X}$ *converge vers* x *alors* $x \in \mathbf{X}$ (exercice)*;*
— *pour tout* $y \in \mathbf{Y}$ *il existe* r *tel que la boule ouverte* $B_r(y)$ *est incluse dans* $\mathbf{Y}$ (exercice) *donc* $\mathbf{Y} = \cup_{y\in\mathbf{Y}} B_r(y)$.

Le théorème de Tychonoff implique que l'espace de Cantor $\mathcal{A}^{\mathbb{Z}^d}$ est compact en tant que produit d'espaces compacts. Comme il s'agit ici d'un produit dénombrable, il est possible de donner une preuve simple en utilisant la caractérisation séquentielle de la compacité.

Proposition 1.2.3. *L'espace de Cantor* $\mathcal{A}^{\mathbb{Z}^d}$ *est compact (pour la topologie produit).*

Démonstration. Comme $\mathcal{A}^{\mathbb{Z}^d}$ est métrisable, il suffit par le théorème de Bolzano-Weierstrass de montrer que toute suite d'éléments de $\mathcal{A}^{\mathbb{Z}^d}$ contient une sous-suite qui converge dans $\mathcal{A}^{\mathbb{Z}^d}$. Soit $(x^n)_{n\in\mathbb{N}}$ une suite de $\mathcal{A}^{\mathbb{Z}^d}$. On extrait de $(x^n)_{n\in\mathbb{N}}$ une sous-suite convergente $(x^{\phi(n)})_{n\in\mathbb{N}}$ grâce au fait qu'il n'existe, pour tout $n \in \mathbb{N}$, qu'un nombre fini de motifs d'un support fixé $\mathbb{U}_n$.

Considérons l'ensemble de tous les symboles pouvant apparaître en position 0 dans les configurations x^n. Cet ensemble est fini, donc par principe des tiroirs il existe un symbole $p_0 \in \mathcal{A}$ qui apparaît infiniment souvent comme un x^n_0. Appelons $\phi(0)$ le plus petit entier n tel que $x^n_0 = p_0$. On construit à présent inductivement $\phi(k)$ à partir de $\phi(k-1)$. En utilisant à nouveau le principe des tiroirs, il existe un motif $p_k \in \mathcal{A}^{\mathbb{U}_k}$ tel que $(p_k)_{|\mathbb{U}_{k-1}} = p_{k-1}$ qui apparaît infiniment souvent comme un $(x^n)_{|\mathbb{U}_k}$. Appelons $\phi(k)$ le plus petit entier $n > \phi(k-1)$ tel que $(x^n)_{|\mathbb{U}_k} = p_k$. Enfin, x sera la configuration dans $\mathcal{A}^{\mathbb{Z}^d}$ donnée par $x_{|\mathbb{U}_k} = p_k$ pour tout $k \in \mathbb{N}$ (x est bien définie car $(p_k)_{k\in\mathbb{N}}$ est une suite croissante de motifs). Reste à montrer que $(x^{\phi(n)})_{n\in\mathbb{N}}$ converge vers x quand n tend vers l'infini, ce qui est bien le cas puisque $d(x, x^{\phi(n)}) \leqslant 2^{-k}$ pour $n \geqslant k$. ■

Cette propriété de compacité de l'ensemble des configurations est fondamentale dans l'étude des sous-décalages. Dans le cas où l'alphabet $\mathcal{A} = \{a_1, a_2, \dots\}$ est infini, on perd la propriété de compacité : la suite $(x^n)_{n\in\mathbb{N}}$ où la configuration $x^n \in \mathcal{A}^{\mathbb{Z}}$ est la configuration uniforme formée du seul symbole a_n ne possède pas de sous-suite convergente.

Exercice 1.2.4. *Soit* $\mathcal{A} = \{0, 1\}$ *et soit* $(x^n)_{n\in\mathbb{N}}$ *la suite de configurations de* $\mathcal{A}^{\mathbb{Z}}$ *où* x^n *contient l'écriture en binaire de* n *avec le bit de poids faible en position* 0 *et qui vaut* 0 *partout ailleurs. Donner explicitement une sous-suite convergente de* $(x^n)_{n\in\mathbb{N}}$.

L'espace $\mathcal{A}^{\mathbb{Z}^d}$ étant muni de sa topologie de Cantor, on a maintenant une notion de fonction continue : $F : \mathcal{A}^{\mathbb{Z}^d} \to \mathcal{A}^{\mathbb{Z}^d}$ est continue en x si

$$\forall \epsilon > 0, \exists \delta, \forall y : d(x,y) \leqslant \delta \Rightarrow d\left(F(x), F(y)\right) \leqslant \epsilon.$$

Informellement, pour connaître le motif fini central de $F(x)$ d'une certaine taille il suffit de connaître le motif central de x d'une certaine taille (éventuellement supérieure).

1.2.3 Sous-décalages

S'il est vu comme un groupe, $\mathbb{Z}^d$ agit de façon continue sur l'ensemble des configurations $\mathcal{A}^{\mathbb{Z}^d}$ par translation, et définit ainsi une dynamique sur cet espace.

Définition 1.2.5. *L'*action du décalage*, ou* décalage*, est définie pour tout élément* $\mathbf{j} \in \mathbb{Z}^d$ *par*

$$\begin{array}{rccc} \sigma^{\mathbf{j}} : & \mathcal{A}^{\mathbb{Z}^d} & \longrightarrow & \mathcal{A}^{\mathbb{Z}^d} \\ & x = (x_{\mathbf{i}})_{\mathbf{i}\in\mathbb{Z}^d} & \longmapsto & \sigma^{\mathbf{j}}(x) = (x_{\mathbf{i}-\mathbf{j}})_{\mathbf{i}\in\mathbb{Z}^d}. \end{array}$$

L'action du décalage est continue (*exercice*).

Définition 1.2.6. *Un* sous-décalage $\mathbf{X}$ *est une partie fermée et* σ*-invariante (c'est-à-dire que* $\sigma^{\mathbf{i}}(\mathbf{X}) \subseteq \mathbf{X}$ *pour tout* $\mathbf{i} \in \mathbb{Z}^d$*) de* $\mathcal{A}^{\mathbb{Z}^d}$.

En particulier, l'ensemble $\mathcal{A}^{\mathbb{Z}^d}$ est appelé *décalage plein* sur l'alphabet $\mathcal{A}$. Avec cette définition, un sous-décalage est donc un sous-système du système dynamique $(\mathcal{A}^{\mathbb{Z}^d}, \sigma)$.

L'*orbite* d'une configuration $x \in \mathcal{A}^{\mathbb{Z}^d}$ est l'ensemble σ-invariant $\mathcal{O}(x) = \{\sigma^{\mathbf{j}}(x) \mid \mathbf{j} \in \mathbb{Z}^d\}$. Une orbite est toujours σ-invariante, mais peut ne pas être fermée comme le montre l'Exemple 1.2.7. La manière classique de définir le *sous-décalage engendré par* x consiste à prendre la clôture topologique de l'orbite de x, c'est-à-dire $\overline{\mathcal{O}(x)}$.

Exemple 1.2.7. *Soit* $x \in \{0,1\}^{\mathbb{Z}^d}$ *la configuration qui ne compte qu'un seul symbole* 1 *en position* $\mathbf{0}$ *: pour tout* $\mathbf{i} \in \mathbb{Z}^d$, $x_{\mathbf{i}} = 1$ *si et seulement si* $\mathbf{i} = \mathbf{0}$. *L'orbite* $\mathcal{O}(x) = \{\sigma^{\mathbf{i}}(x) \mid \mathbf{i} \in \mathbb{Z}^d\}$, *qui est* σ*-invariante, est l'ensemble de toutes les configurations dans* $\{0,1\}^{\mathbb{Z}^d}$ *qui contiennent exactement un symbole* 1. *Cependant* $\mathcal{O}(x) = \{\sigma^{\mathbf{i}}(x) \mid \mathbf{i} \in \mathbb{Z}^d\}$ *n'est pas fermé, car la suite* $(\sigma^{n\mathbf{k}}(x))_{n\in\mathbb{N}}$ *pour un certain* $\mathbf{k} \in \mathbb{Z}^d$, $\mathbf{k} \neq \mathbf{0}$, *converge vers la configuration uniforme* $0^{\mathbb{Z}^d}$. *Le sous-décalage engendré par* x *est*

$$\mathbf{X}_{\leqslant 1} = \overline{\mathcal{O}(x)} = \{\sigma^{\mathbf{i}}(x) \mid \mathbf{i} \in \mathbb{Z}^d\} \cup \{0^{\mathbb{Z}^d}\}.$$

Soit $\mathbf{X}$ un sous-décalage. Le *langage* de $\mathbf{X}$, noté $\mathcal{L}(\mathbf{X})$, est l'ensemble de tous les motifs qui apparaissent dans au moins une configuration de $\mathbf{X}$. Le *langage de taille n*, noté $\mathcal{L}_n(\mathbf{X})$, est l'ensemble de tous les motifs de support élémentaire $\mathbb{U}_n = [-n, n]^d$ de taille n qui apparaissent dans au moins une configuration de $\mathbf{X}$:

$$\mathcal{L}_n(\mathbf{X}) = \left\{ p \in \mathcal{A}^{\mathbb{U}_n} \mid p \text{ apparaît dans une configuration de } \mathbf{X} \right\}.$$

Le *langage élémentaire* de $\mathbf{X}$, noté $\mathcal{L}_e(\mathbf{X})$, est l'ensemble de tous les motifs élémentaires qui apparaissent dans une configuration de $\mathbf{X}$:

$$\mathcal{L}_e(\mathbf{X}) = \bigcup_{n \in \mathbb{N}} \mathcal{L}_n(\mathbf{X}).$$

N'importe quel ensemble de motifs ne peut pas être le langage d'un certain sous-décalage, comme la caractérisation suivante le montre.

Proposition 1.2.8. *Soit $\mathcal{L}$ un ensemble de motifs. Alors $\mathcal{L}$ est le langage d'un sous-décalage si et seulement les deux conditions suivantes sont respectées :*

- *$\mathcal{L}$ est* factoriel *: $\forall p \in \mathcal{L}$, si $p' \sqsubseteq p$ alors $p' \in \mathcal{L}$;*
- *$\mathcal{L}$ est* extensible *: $\forall p \in \mathcal{L}, \exists p' \in \mathcal{L}$, avec $\mathrm{supp}(p') = \mathrm{supp}(p) + [-1, 1]^d$ qui vérifie $p \sqsubset p'$.*

Démonstration. Soit $p \in \mathcal{L}(\mathbf{X})$, alors il existe $x \in \mathbf{X}$ tel que $p \sqsubset_{\mathbf{0}} x$. Par conséquent, pour tout motif $p' \sqsubset p$, on a $p' \sqsubset x$ donc $\mathcal{L}(\mathbf{X})$ est factoriel. De plus $x_{\mathrm{supp}(p)+[-1,1]^d}$ est un élément de $\mathcal{L}(\mathbf{X})$ qui contient p, donc $\mathcal{L}(\mathbf{X})$ est aussi extensible.

Supposons maintenant que $\mathcal{L}$ est un ensemble de motifs sur $\mathcal{A}$ qui soit à la fois factoriel et extensible. Considérons l'ensemble $\mathbf{X} = \{x \in \mathcal{A}^{\mathbb{Z}^d} \mid$ tout motif de x est dans $\mathcal{L}\}$, et montrons que $\mathbf{X}$ est bien un sous-décalage. Clairement $\mathbf{X}$ est σ-invariant. Soit une suite $(x^n)_{n \in \mathbb{N}}$ d'éléments de $\mathbf{X}$ qui converge vers x. Supposons qu'un motif p de x ne soit pas dans $\mathcal{L}$. Alors il existe $\mathbb{U} \subset \mathbb{Z}^d$ fini et $N \in \mathbb{N}$ tels que $x^n_{\mathbb{U}}$ est constante pour tout $n \geqslant N$ et $p \sqsubset x^n_{\mathbb{U}}$. Ceci contredit le fait que $(x^n)_{n \in \mathbb{N}}$ est une suite d'éléments de $\mathbf{X}$. Ainsi, $\mathbf{X}$ est fermé et donc un sous-décalage.

Par construction, on a que $\mathcal{L}(\mathbf{X}) \subset \mathcal{L}$. Réciproquement, par compacité tous les motifs de $\mathcal{L}$ peuvent être prolongés en une configuration de $\mathbf{X}$, donc $\mathcal{L} \subset \mathcal{L}(\mathbf{X})$. ■

Définition 1.2.9. *Soit $\mathcal{F}$ un ensemble de motifs sur l'alphabet $\mathcal{A}$. Le* sous-décalage défini par $\mathcal{F}$ *est l'ensemble des configurations qui évitent les motifs de $\mathcal{F}$:*

$$\mathbf{X}_{\mathcal{F}} = \left\{ x \in \mathcal{A}^{\mathbb{Z}^d} \mid p \text{ n'apparaît pas dans } x \text{ pour tout } p \in \mathcal{F} \right\}.$$

On peut réécrire la définition précédente à l'aide des cylindres. On a ainsi :

$$\mathbf{X}_{\mathcal{F}} = \mathcal{A}^{\mathbb{Z}^d} \setminus \bigcup_{p\in\mathcal{F}|\mathbf{i}\in\mathbb{Z}^d} \sigma^{-\mathbf{i}}([\![p]\!]).$$

Par définition $\mathbf{X}_{\mathcal{F}}$ est σ-invariant ; il est aussi fermé d'après ce qui précède, car c'est le complémentaire d'une union de cylindres.

Remarque 1.2.10. *Pour montrer que le sous-décalage $\mathbf{X}_{\mathcal{F}}$ n'est pas vide, il suffit de montrer que pour tout $n \in \mathbb{N}$ il existe un motif de support $[-n,n]^d$ qui ne contient aucun motif de $\mathcal{F}$ (*exercice*).*

Le langage $\mathcal{L}(\mathbf{X})$ d'un sous-décalage $\mathbf{X}$ est formé de l'ensemble de tous les motifs *autorisés* (ceux qui apparaissent dans au moins une configuration). Son complémentaire, qu'on notera $\mathcal{L}(\mathbf{X})^c$, est donc précisément l'ensemble de tous les motifs interdits, autrement dit le plus grand ensemble (pour l'inclusion) de motifs interdits qui définit $\mathbf{X}$. En fait, tout sous-décalage peut s'écrire de la sorte, comme précisé dans la proposition qui suit.

Proposition 1.2.11. *Soit $\mathbf{X}$ un sous-décalage. Alors le complémentaire de son langage $\mathcal{L}(\mathbf{X})^c$ est un ensemble de motifs interdits qui le définit, c'est-à-dire que $\mathbf{X} = \mathbf{X}_{\mathcal{L}(\mathbf{X})^c}$.*

Démonstration. Soit x une configuration du sous-décalage $\mathbf{X}$. Comme le langage $\mathcal{L}(\mathbf{X})$ contient par définition tous les motifs qui apparaissent dans une configuration de $\mathbf{X}$, tous les motifs apparaissant dans x s'y trouvent. Ainsi x appartient au sous-décalage défini par l'ensemble de motifs interdits $\mathcal{L}(\mathbf{X})^c$.

Réciproquement, supposons que $x \in \mathbf{X}_{\mathcal{L}(\mathbf{X})^c}$. Alors tout motif qui apparaît dans x est aussi dans $\mathcal{L}(\mathbf{X})$. En particulier pour tout $\mathbb{U} \subset \mathbb{Z}^d$, le motif $x_{\mathbb{U}}$ appartient au langage de $\mathbf{X}$. Par conséquent, il existe une configuration x^n dans $\mathbf{X}$ qui coïncide avec x sur le support $\mathbb{U}_n = [-n,n]^d$, autrement dit $(x^n)_{\mathbb{U}_n} = x_{\mathbb{U}_n}$ pour tout $n \in \mathbb{N}$. On peut construire une configuration $x^n \in \mathbf{X}$ pour tout entier n, de sorte que la suite de configurations $(x^n)_{n\in\mathbb{N}}$ converge vers x. Le sous-décalage $\mathbf{X}$ étant fermé, on en déduit que x appartient à $\mathbf{X}$. ■

Remarque 1.2.12. *La notion de langage d'un sous-décalage est tout à fait naturelle, et pour un sous-décalage donné, cet ensemble de motifs est unique. Malheureusement, un langage de sous-décalage est aussi un objet très contraint, comme le montre la Proposition 1.2.8. C'est la raison pour laquelle on définit habituellement un sous-décalage par un ensemble de motifs interdits plutôt que par son langage (voir exemple 1.2.14).*

On a donc bien deux définitions équivalentes d'un même objet : l'une topologique (Définition 1.2.6), où le sous-décalage est vu dans son ensemble, l'autre combinatoire (Définition 1.2.9), où les configurations du sous-décalages sont décrites par les contraintes locales qu'elles respectent.

On introduit maintenant la classe des sous-décalages de type fini, dont les configurations vérifient un nombre fini de contraintes locales. Ce sont les décalages les plus simples à définir et les plus aisément manipulables, et ils permettent aussi de modéliser des phénomènes réels décrits de manière locale. Les automates cellulaires, qui seront introduits dans la Partie 1.2.4, sont un exemple de systèmes dynamiques qui peuvent s'interpréter en terme de sous-décalages.

Définition 1.2.13. *Un sous-décalage est dit* de type fini *si on peut le définir par un ensemble fini de motifs interdits.*

Par souci de concision, on appellera dans la suite *SFT* (de l'anglais *shift of finite type*) un sous-décalage de type fini.

Si $\mathbf{X}$ est un SFT donné par l'ensemble $\mathcal{F}$ de motifs interdits, on peut supposer que tous les motifs de $\mathcal{F}$ ont le même support élémentaire (*exercice*).

Exemple 1.2.14. *En dimension* 1, *le SFT défini par l'unique motif interdit* ■□ *sur l'alphabet* $\mathcal{A} = \{\square, \blacksquare\}$ *ne contient que trois types de configurations : les configurations uniformes* $\square^{\mathbb{Z}}$ *et* $\blacksquare^{\mathbb{Z}}$, *ainsi que la configuration* x *telle que* $x_i = \blacksquare$ *si et seulement si* $i > 0$ *et toutes ses translatées.*

Exemple 1.2.15. *Le sous-décalage* $\mathbf{X}_{\leqslant 1}$ *défini dans l'Exemple 1.2.7 n'est pas de type fini* (exercice).

Exemple 1.2.16. *Une tuile de Wang est un carré portant une couleur sur chacun de ses côtés. Étant donné un ensemble fini de tuiles de Wang* τ, *on définit le SFT* $\mathbf{X}_\tau \subset \tau^{\mathbb{Z}^2}$ *par les contraintes locales suivantes : si* t *et* t' *sont deux tuiles voisines, alors elles portent la même couleur sur leur côté commun.*

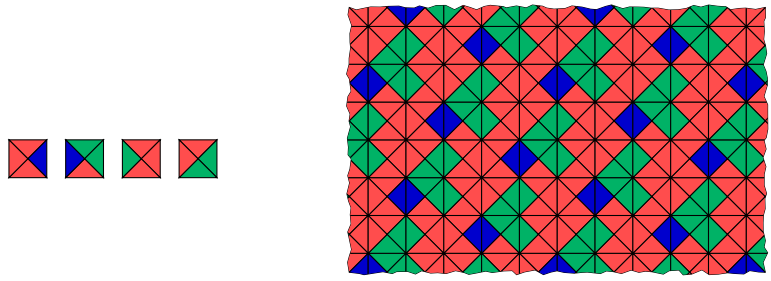


FIGURE 1.2 – *Un jeu de tuiles de Wang* τ *et un exemple de pavage par* τ.

Les ensembles de pavages par tuiles de Wang sont des exemples de SFT, et comme nous le verrons plus loin (Partie 1.2.5), tout SFT peut se recoder dans un jeu fini de tuiles de Wang.

1.2.4 Automates cellulaires

Un automate cellulaire (déterministe) est une fonction agissant sur $\mathcal{A}^{\mathbb{Z}^d}$ en changeant le symbole en chaque position de façon synchrone, uniforme et locale (*i.e.* seulement en fonction du motif apparaissant dans un voisinage fini). La métaphore est celle d'un réseau de cellules (positions de $\mathbb{Z}^d$) contenant chacune un petit automate qui met à jour son état en observant les cellules voisines.

Définition 1.2.17. *Un* automate cellulaire (AC) *sur l'espace $\mathcal{A}^{\mathbb{Z}^d}$ est une fonction $F : \mathcal{A}^{\mathbb{Z}^d} \to \mathcal{A}^{\mathbb{Z}^d}$ définie par un ensemble fini $\mathbb{U} \subseteq \mathbb{Z}^d$ appelé* voisinage *et une* fonction de transition locale *$f : \mathcal{A}^{\mathbb{U}} \to \mathcal{A}$ de la façon suivante :*

$$\forall \mathbf{i} \in \mathbb{Z}^d : F(x)(\mathbf{i}) = f(x_{|\mathbf{i}+\mathbb{U}})$$

où $x_{|\mathbf{i}+\mathbb{U}}$ est la fonction $\mathbf{j} \in \mathbb{U} \mapsto x(\mathbf{i}+\mathbf{j})$.

Exemple. En dimension 1, avec $\mathcal{A} = \{0,1\}$, $\mathbb{U} = \{-1,0\}$ et f telle que $f(a,b,c) = a + b \bmod 2$, on a l'automate cellulaire $F : \mathcal{A}^{\mathbb{Z}^d} \to \mathcal{A}^{\mathbb{Z}^d}$ suivant :

$$F(x)(\mathbf{i}) = x_{\mathbf{i}-1} + x_{\mathbf{i}} \bmod 2$$

$F^8(x)$	.....	0	1	0	0	0	0	0	0	0 .....
$F^7(x)$	.....	0	1	1	1	1	1	1	1	1 .....
$F^6(x)$	.....	0	1	0	1	0	1	0	1	0 .....
$F^5(x)$	.....	0	1	1	0	0	1	1	0	0 .....
$F^4(x)$	.....	0	1	0	0	0	1	0	0	0 .....
$F^3(x)$	.....	0	1	1	1	1	0	0	0	0 .....
$F^2(x)$	.....	0	1	0	1	0	0	0	0	0 .....
$F(x)$	.....	0	1	1	0	0	0	0	0	0 .....
x	.....	0	1	0	0	0	0	0	0	0 .....

Dans la définition ci-dessus, un automate cellulaire est en fait deux choses à la fois :

— un *système dynamique*, *i.e.* une fonction F agissant globalement sur les configurations infinies ;

— une *description finie et locale* de ce système donnée par $(\mathcal{A}, \mathbb{U}, f)$.

Avoir une description finie est utile lorsqu'on s'intéresse à des problèmes de décision, mais on peut également ne pas l'expliciter et définir les automates cellulaires de manière topologique comme le montre le théorème suivant.

Théorème 1.2.18 (Curtis-Lyondon-Hedlund [44]). *Une fonction F de $\mathcal{A}^{\mathbb{Z}^d}$ dans lui-même est un automate cellulaire si et seulement si elle satisfait les deux conditions suivantes :*

1. *elle est continue ;*
2. $\sigma^{\mathbf{i}} \circ F = F \circ \sigma^{\mathbf{i}}$ *pour tout* $\mathbf{i} \in \mathbb{Z}^d$.

Démonstration. Tout d'abord un automate cellulaire défini par

$$\forall \mathbf{j} \in \mathbb{Z}^d : F(x)(\mathbf{j}) = f(x_{|\mathbf{j}+\mathbb{U}})$$

commute avec les translations (car cette définition est uniforme en $\mathbf{j}$) et est continue car $F(x)$ et $F(y)$ coïncident sur toutes les positions $|\mathbf{i}| \leqslant n$ dès lors que x et y coïncident sur toutes les positions $|\mathbf{i}| \leqslant n + C$ où $C = \max\{|\mathbf{j}| \mid \mathbf{j} \in \mathbb{U}\}$, autrement dit :

$$\forall \epsilon \geqslant 0, \forall x, y \in \mathcal{A}^{\mathbb{Z}^d} : d(x,y) \leqslant 2^{-C}\epsilon \Rightarrow d\left(F(x), F(y)\right) \leqslant \epsilon.$$

Réciproquement, soit F une fonction continue qui commute avec les translations. Comme F est continue sur un compact, elle est uniformément continue (théorème de Heine) :

$$\forall \epsilon \geqslant 0, \exists \delta, \forall x, y \in \mathcal{A}^{\mathbb{Z}^d} : d(x,y) \leqslant \delta \Rightarrow d\left(F(x), F(y)\right) \leqslant \epsilon.$$

Choisissons $\epsilon = \frac{1}{2}$ et soit $\delta_{\frac{1}{2}} \leqslant 1$ une valeur de la constante δ correspondante garantie par la continuité uniforme. On a alors que $F(x)(\vec{0})$ (où $\vec{0}$ est l'origine de $\mathbb{Z}^d$) est une fonction de $x_{|\mathbb{U}}$ où $\mathbb{U} = \{\mathbf{i} \mid |\mathbf{i}| \leqslant -\log(\delta_{\frac{1}{2}})\}$, *i.e.* $F(x)(\vec{0}) = f(x_{|\mathbb{U}})$ pour une certaine fonction $f : \mathcal{A}^{\mathbb{U}} \to \mathcal{A}$. Mais F commute avec les translations, on a donc finalement $F(x)(\mathbf{j}) = f(x_{|\mathbf{j}+\mathbb{U}})$ pour tout $\mathbf{j} \in \mathbb{Z}^d$. ■

Corollaire 1.2.19. *Si un automate cellulaire F est une fonction bijective, alors son inverse F^{-1} est aussi un automate cellulaire. Dans ce cas, on dit que F est* réversible.

Démonstration. L'inverse d'une fonction continue bijective sur un compact est une fonction continue. Par ailleurs, de l'égalité $\sigma^{\mathbf{i}} \circ F = F \circ \sigma^{\mathbf{i}}$ on déduit l'égalité $F^{-1} \circ \sigma^{\mathbf{i}} = \sigma^{\mathbf{i}} \circ F^{-1}$. Le théorème 1.2.18 permet de conclure. ■

La preuve ci-dessus nous assure l'existence d'un inverse de façon non constructive en utilisant seulement des propriétés topologiques de l'espace $\mathcal{A}^{\mathbb{Z}^d}$. On peut bien entendu se demander si on peut construire *algorithmiquement* cet inverse et si on a des bornes sur son rayon. La section 1.5 nous fournira (après beaucoup de travail) une réponse étonnante : ça dépend de la dimension !

Un automate cellulaire bijectif est réversible, mais en fait la propriété d'injectivité est suffisante.

Proposition 1.2.20. *Tout automate cellulaire injectif est surjectif et donc bijectif et réversible.*

Démonstration. Considérons l'ensemble X_n des configurations périodiques de période n dans toutes les directions de l'espace. On a $F(X_n) \subseteq X_n$ pour tout AC (par commutation avec les translation). Mais si F est injectif, sa restriction à X_n est injective et donc surjective car X_n est un ensemble fini. On vient donc de montrer que pour tout n et tout $x_n \in X_n$ il existe y_n telle que $F(y_n) = x_n$. Considérons à présent n'importe quelle configuration x. On peut l'approcher par une suite de configurations (x_n) telle que $x_n \in X_n$ pour tout n et $x_n \xrightarrow[n\to\infty]{} x$. D'après ce qui précède il existe donc une suite de configurations (y_n) telle que $F(y_n) \xrightarrow[n\to\infty]{} x$. Par compacité de l'espace, on peut en extraire une sous suite convergente $y_{n_p} \xrightarrow[p\to\infty]{} y$. Par continuité de F on a alors $F(y) = x$ et on en déduit que F est surjectif. ■

Soit F un AC et $I_1 = F(\mathcal{A}^{\mathbb{Z}^d})$. I_1 est un sous-décalage car invariant par translation (F commute avec les translations) et fermé (image d'un fermé par une application continue sur un compact). F est surjectif si et seulement si $I_1 = \mathcal{A}^{\mathbb{Z}^d}$.

Comme vu plus haut, l'injectivité implique la surjectivité. En fait on peut caractériser la surjectivité par une forme faible d'injectivité et au passage fournir une autre preuve (plus longue !) de la proposition 1.2.20. Soient $c, d \in \mathcal{A}^{\mathbb{Z}^d}$. On dit que d est *c-finie* si elle ne diffère de c qu'en un nombre fini de points de $\mathbb{Z}^d$, *i.e.* si

$$\{z \in \mathbb{Z}^d \mid c(z) \neq d(z)\}$$

est fini. F est *pré-injectif* si, pour toute paire c, d avec d une configuration c-finie, $F(c) = F(d)$ implique $c = d$. Un des plus vieux théorèmes de la théorie des AC est le suivant.

Théorème 1.2.21 (Moore-Myhill [90, 97])**.** *Un AC est surjectif si et seulement si il est pré-injectif.*

FIGURE 1.3 – *Diagramme espace-temps d'un AC nilpotent 1D à 3 états et de voisinage* $\{-1, 0, 1\}$ *: les lignes successives du diagramme représentent (du bas vers le haut) les configurations successives obtenues par application de l'AC.*

Pour tout AC F on a $F(I_1) \subseteq I_1$ et plus plus généralement $I_{n+1} \subseteq I_n$ où $I_n = F^n(\mathcal{A}^{\mathbb{Z}^d})$. Tous ces ensembles sont des sous-décalages et leur limite

$$\Omega_F = \bigcap_{n \geqslant 1} I_n$$

est appelée *ensemble oméga-limite de F* (une notion très étudiée dans la théorie générale des systèmes dynamiques). Ω_F est toujours non-vide : il contient toujours au moins une configuration uniforme (*i.e.* où toutes les cellules sont dans le même état). Ω_F est l'ensemble des configurations $x \in \mathcal{A}^{\mathbb{Z}^d}$ telles qu'il existe une suite (x_n) avec $x_0 = x$ et $x_n = F(x_{n+1})$ pour tout n, c'est-à-dire celles qui ont une "histoire infinie". En effet, avoir une "histoire infinie" implique l'appartenance à chacun des I_n donc à Ω_F. Réciproquement, si $x \in \Omega_F$ alors pour tout n il y a y_n telle que $F^n(y_n) = x$. Soit $z_n = F^n(y_{n+1})$. On a, pour tout n, $z_n \in I_n$ et $F(z_n) = x$. Soit z une valeur d'adhérence de la suite $(z_n)_n$ (qui existe par compacité de $\mathcal{A}^{\mathbb{Z}^d}$), on a alors : $z \in I_n$ pour tout n (car I_n est un fermé), donc $z \in \Omega_F$, et $F(z) = x$ (par continuité de F). Ainsi chaque configuration de Ω_F a un antécédent dans Ω_F et donc une histoire infinie.

Un AC est *nilpotent* si Ω_F contient une seule configuration (nécessairement uniforme car Ω_F contient toujours une configuration uniforme). Cette propriété correspond à un comportement extrême d'irréversibilité : de moins en moins de motifs peuvent apparaître au cours du temps jusqu'à arriver en temps fini (voir l'exercice suivant) à une même configuration uniforme. En effet, on peut montrer que si F est nilpotent alors il existe $t \geqslant 1$ tel que F^t est une fonction constante qui donne toujours la même configuration uniforme (*exercice*).

1.2.5 Liens entre sous-décalages et automates cellulaires

Il a été remarqué plus haut que $I_1 = F(\mathcal{A}^{\mathbb{Z}^d})$ est un sous-décalage pour tout AC F. Plus généralement, l'image de tout sous-décalage par un AC est un sous-décalage (*exercice*). Cela donne lieu à deux notions importantes en dynamique symbolique :

1. un sous-décalage qui est l'image d'un SFT (*i.e.* sous-décalage de type fini) par un automate cellulaire est dit *sofique* [1] [135] : $\mathbf{X}$ est un sous-décalage sofique s'il s'écrit $\mathbf{X} = F(\mathbf{Y})$ où F est un AC et $\mathbf{Y}$ un SFT ;
2. deux sous-décalages $\mathbf{X}$ et $\mathbf{Y}$ de $\mathcal{A}^{\mathbb{Z}^d}$ sont dits *conjugués* s'il existe des automates cellulaires F et G tels que $F(\mathbf{X}) = \mathbf{Y}$ et $G(\mathbf{Y}) = \mathbf{X}$.

Exemple 1.2.22. *Le sous-décalage $\mathbf{X}_{\leqslant 1}$ défini dans l'exemple 1.2.7 est sofique. En effet, si on considère le SFT $\mathbf{X}$ sur l'alphabet $\{0,1\}$ défini par le motif interdit 10 (c'est exactement le SFT de l'exemple 1.2.14 en codant blanc par 0 et noir par 1) et l'AC F de voisinage $\mathbb{U} = \{0,1\}$ et de fonction locale*

$$f(a,b) = \begin{cases} 1 & \text{si } ab = 01 \\ 0 & \text{sinon} \end{cases}$$

*alors on a $\mathbf{X}_{\leqslant 1} = F(\mathbf{X})$ (*exercice*).*

Remarque 1.2.23. *Pour alléger les notations, on suppose que les sous-décalages partagent un même alphabet. Ce n'est pas une contrainte (on peut toujours voir deux sous-décalages sur des alphabets distincts comme des sous-décalage de $\mathcal{A}^{\mathbb{Z}^d}$ où $\mathcal{A}$ est l'union de leurs alphabets) mais il faut garder à l'esprit qu'un sous-décalage utilisant seulement deux symboles peut être l'image d'un sous-décalage avec beaucoup plus de symboles.*

Exemple 1.2.24. *Un SFT est toujours conjugué à un ensemble de pavages par tuiles de Wang (*exercice*). En conséquence, lorsqu'on s'intéresse à des problèmes invariants par conjugaison, on utilise indifféremment des SFT ou des ensembles de pavages par tuiles de Wang.*

Proposition 1.2.25 (voir [81])**.** *On a les propriétés suivantes :*

- *si $\mathbf{X}$ est un sous-décalage sofique et F un AC alors $F(\mathbf{X})$ est sofique ;*
- *l'union de deux sous-décalages sofique est un sous-décalage sofique ;*
- *un sous-décalage conjugué à un SFT est un SFT.*

1. Le mot "sofique" (en anglais *sofic*) vient du mot "fini" en hébreu. Il apparaît également en mathématiques dans la notion de "groupe sofique" (qui n'a *a priori* aucun lien avec la notion de sous-décalage sofique).

En particulier, tous les sous-décalages I_n définis en sous-section 1.2.4 sont sofiques. Ω_F peut lui aussi être sofique (comme pour les AC nilpotents), mais ce n'est pas toujours le cas [25].

Exercice 1.2.26. *Soit $\mathcal{A} = \{0, 1\}$. Soit $\mathbf{X}_{01}$ (resp. $\mathbf{X}_{10}$) le sous-décalage de $\mathcal{A}^{\mathbb{Z}}$ défini par le motif interdit* 10 *(resp.* 01*). Montrer que $\mathbf{X}_{01} \cup \mathbf{X}_{10}$ n'est pas un SFT, mais que c'est un sous-décalage sofique.*

Les SFT peuvent aussi être caractérisés par automate cellulaire. Si F est un AC , on note $\Phi_F = \{c \in \mathcal{A}^{\mathbb{Z}^d} \mid c = F(c)\}$ l'ensemble de ses points fixes. Φ_F est toujours un SFT (*exercice*). Réciproquement, pour tout SFT $\mathbf{X}$ il existe un AC F tel que $\mathbf{X} = \Phi_F$: il suffit de considérer un F qui change (arbitrairement) la valeur d'une cellule si et seulement si un motif interdit apparaît dans son voisinage (*exercice*).

Un autre lien important entre automates cellulaires et sous-décalages passe par la notion de *diagramme espace-temps* : configuration de dimension $d+1$ qui représente une succession de configurations de dimension d apparaissant dans l'évolution d'un automate cellulaire donné, « empilées » les unes sur les autres (voir la figure 1.3). Ce point de vue géométrique sur les systèmes dynamiques que sont les automates cellulaires s'est révélé fécond, notamment pour prouver des résultats négatifs (voir section 1.5).

1.3 Apériodicité

1.3.1 Périodicité, apériodicité et conjecture de Wang

Soit $x \in \mathcal{A}^{\mathbb{Z}^d}$ une configuration. On dit que $\mathbf{i} \in \mathbb{Z}^d$ non nul est une *direction de périodicité* pour x si $\sigma^{\mathbf{i}}(x) = x$, autrement dit la translation selon $\mathbf{i}$ laisse x invariante. On dit qu'une configuration $x \in \mathcal{A}^{\mathbb{Z}^d}$ est *périodique* si x possède d directions de périodicité linéairement indépendantes. En dimension d, une configuration périodique peut donc être entièrement décrite par un motif fini répété selon un sous-groupe de $\mathbb{Z}^d$.

Voici quelques exemples qui illustrent les propriétés des sous-décalages possédant une configuration périodique.

1. En dimension 1, tout SFT non vide possède une configuration périodique (*exercice*) ;
2. si $\mathbf{X}$ et $\mathbf{X}'$ sont deux sous-décalages conjugués, alors $\mathbf{X}$ possède une configuration périodique si et seulement si $\mathbf{X}'$ en possède une (*exercice*) ;

3. en dimension d, si un SFT contient une configuration avec $d-1$ directions de périodicité linéairement indépendantes, alors il contient aussi une configuration périodique (*exercice*).

On dit qu'un SFT (resp. un jeu de tuiles de Wang, voir exemple 1.2.16) est *apériodique* s'il contient au moins une configuration (resp. permet de paver le plan) et qu'aucune de ses configurations (resp. pavages) n'est périodique. Dans son article introduisant le modèle des tuiles de Wang [134], ce dernier conjecture qu'il n'existe aucun jeu de tuiles apériodique.

Conjecture 1.3.1 (Wang (1961))**.** *Si un jeu de tuiles permet de paver le plan, alors il peut le faire de manière périodique.*

Nous allons voir dans la suite comment cette conjecture sera réfutée par Berger [8], et en quoi elle est en lien avec la décidabilité du problème du Domino (voir la Partie 1.5.2).

1.3.2 Jeu de tuiles de Robinson

En 1966, Berger [8] exhibe un premier exemple de jeu de tuiles de Wang apériodique, réfutant ainsi l'hypothèse formulée par Wang. Nous présentons ici un autre exemple, beaucoup plus simple, dû à Robinson [112].

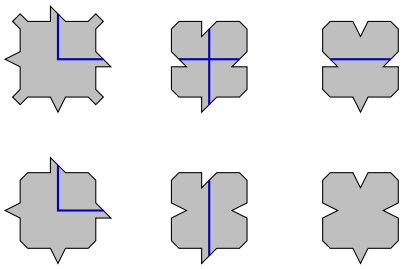


FIGURE *1.4 – Le jeu de tuiles de Robinson, pour lesquelles on autorise les rotations et réflexions.*

Théorème 1.3.2 (Robinson [112])**.** *Le jeu de tuiles de Robinson est apériodique.*

Ce qui suit ne constitue pas une preuve, mais donne les éléments clés de compréhension (se référer à [112] pour plus de détails). Dans la Figure 1.4, la tuile en haut à gauche est dite *bumpy*, toutes les autres sont *dented*. Les encoches triangulaires présentes au centre des côtés sont appelées *flèches*.

Pour montrer l'existence d'un pavage valide par ce jeu de tuiles, on met en évidence la possibilité de construire une structure hiérarchique de

carrés imbriqués, ces carrés pouvant atteindre une taille arbitrairement grande. Une fois encore, la compacité nous permet de conclure quant à l'existence d'un pavage du plan entier.

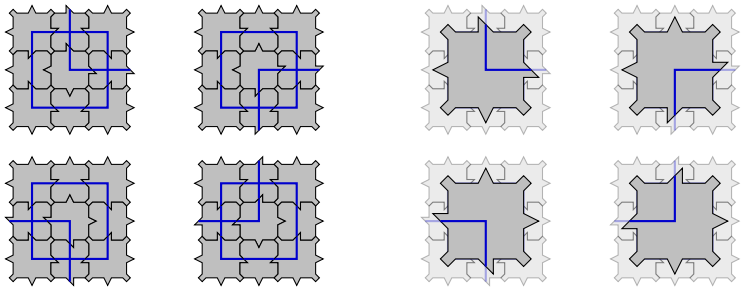


FIGURE *1.5 – Les macrotuiles de niveau 1, qui se comportent comme des tuiles* bumpy.

Plus précisément, on appellera *macro-tuile* de niveau 0 une tuile *bumpy*. Une macro-tuile de niveau $n+1$ est formée de quatre macro-tuiles de niveau n dont les coins se font face, placées autour d'une croix portant en son centre un coin *dented*. Les autres tuiles formant la croix sont déterminées de manière unique par l'orientation choisie pour le coin *dented* (voir la Figure 1.5 pour la construction des macrotuiles de niveau 1).

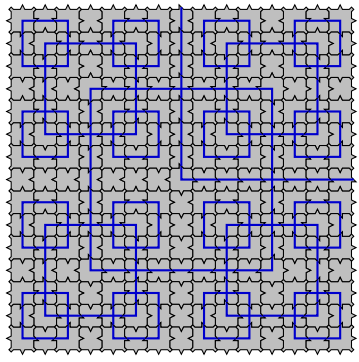


FIGURE *1.6 – Exemple de motif fini dans un pavage par le jeu de Robinson.*

On peut montrer que les pavages obtenus en combinant des macro-tuiles tel qu'expliqué plus haut sont les seuls possibles (voir la Figure 1.6). Un pavage produit par le jeu de Robinson contient donc une hiérarchie infinie de carrés : un carré de niveau $n+1$ est formé à partir de quatre carrés de niveau n, chacun de ces derniers portant en son centre un coin du carré de niveau $n+1$. L'apériodicité découle naturellement : une période

supposée doit envoyer des carrés sur des carrés de même taille. Mais comme il existe des carrés de taille arbitraire, on peut nécessairement trouver un carré que son image par la période supposée intersecte, ce qui contredit l'existence d'une période.

1.3.3 Jeu de tuiles de Kari-Culik

Il existe d'autres façon de produire de l'apériodicité dans les pavages. Pour les tuiles de Wang, l'autre manière classique est due à Kari et Culik [62, 24]. Il s'agit d'encoder un système dynamique très simple : une fonction affine par morceaux sur un intervalle fermé, avec paramètres rationnels bien choisis de sorte que le système est apériodique (en partant d'un point quelconque et en itérant la fonction affine par morceaux, on ne retombe jamais sur ce même point). L'apériodicité du jeu de tuiles provient directement de l'apériodicité du système dynamique. Avec cette technique, on peut obtenir un jeu de tuiles de Wang apériodique à seulement 13 tuiles [24]. Très récemment, un jeu de seulement 11 tuiles a été exhibé, et il a été prouvé que 11 est le plus petit nombre de tuiles requises pour forcer l'apériodicité [51].

1.4 Machines de Turing et décidabilité

Les machines de Turing ont été imaginées par Turing en 1936 [132], afin de délimiter ce qu'un d'algorithme est capable de faire ou non. Il est communément admis que ce qui est effectivement calculable l'est par une machine de Turing (cet énoncé constitue la thèse de Church). Dans cette partie nous rappelons la définition des machines de Turing ainsi que certaines de leurs propriétés fondamentales. Nous montrons aussi en quoi les décalages de type fini constituent un bon moyen de visualiser l'évolution des machines de Turing, à l'aide des diagrammes espace-temps de ces dernières.

1.4.1 Un modèle de calcul robuste

Définition 1.4.1. *Une machine de Turing (MT) est la donnée de* $\mathcal{M} = (Q, \Gamma, \sharp, q_0, \delta, Q_F)$ *où :*

- Q *est un l'ensemble fini des états possibles pour la tête de calcul ;* $q_0 \in Q$ *est l'état initial ;*
- Γ *est un alphabet fini ;*
- $\sharp \notin \Gamma$ *est le symbole blanc ;*

— $\delta : Q \times \Gamma \to Q \times \Gamma \times \{\leftarrow, \cdot, \rightarrow\}$ *est la fonction de transition. Étant donnés un état de la tête de calcul et la lettre inscrite sur le ruban à la position de la tête de calcul, cette lettre est remplacée par une nouvelle lettre, la tête de calcul se déplace sur une cellule adjacente (ou bien reste en place) et passe dans un nouvel état ;*
— $F \subset Q_F$ *est l'ensemble des états finaux. Lorsqu'un état final est atteint, la machine arrête son calcul.*

Exemple 1.4.2. *Considérons la machine de Turing suivante $\mathcal{M}$, qui travaille sur l'alphabet à deux lettres $\Gamma = \{0, 1\}$ et dont la tête de calcul peut être dans trois états $Q = \{q_0, q_+, q_f\}$. Les règles de transitions sont synthétisées dans le tableau suivant :*

$\delta(q, x)$		*Symbole x*		
		0	1	♯
État q	q_0	$(q_0, 0, \rightarrow)$	$(q_0, 1, \rightarrow)$	$(q_+, \sharp, \leftarrow)$
	q_+	$(q_f, 1, .)$	$(q_+, 0, \leftarrow)$	$(q_f, 1, .)$
	q_f	$(q_f, 0, .)$	$(q_f, 1, .)$	$(q_f, \sharp, .)$

Lorsque la tête de calcul est dans l'état initial q_0, elle va chercher le premier symbole ♯ vers la droite, puis passe dans l'état q_+. Une fois dans l'état q_+, la tête de lecture repart vers la gauche en remplaçant les symboles 1 *par des symboles* 0*. Dès qu'un symbole* 0 *ou ♯ est rencontré, il est remplacé par un* 1 *et la machine entre dans son état final. On voit donc que cette machine ajoute* 1 *au nombre codé en binaire sur son ruban d'entrée.*

Un calcul de cette machine sur l'entrée 10011 *passera nécessairement par les configurations du ruban de la Figure 1.7 (où le temps s'écoule de bas en haut).*

...	...	...	...	...	...	...	...	...	...
...	♯	1	0	$(q_f, 1)$	0	0	♯	♯	...
...	♯	1	0	$(q_f, 1)$	0	0	♯	♯	...
...	♯	1	0	$(q_f, 1)$	0	0	♯	♯	...
...	♯	1	0	$(q_+, 0)$	0	0	♯	♯	...
...	♯	1	0	0	$(q_+, 1)$	0	♯	♯	...
...	♯	1	0	0	1	$(q_+, 1)$	♯	♯	...
...	♯	1	0	0	1	1	$(q_0, \sharp)$	♯	...
...	♯	1	0	0	1	$(q_0, 1)$	♯	♯	...
...	♯	1	0	0	$(q_0, 1)$	1	♯	♯	...
...	♯	1	0	$(q_0, 0)$	1	1	♯	♯	...
...	♯	1	$(q_0, 0)$	0	1	1	♯	♯	...
...	♯	$(q_0, 1)$	0	0	1	1	♯	♯	...

FIGURE 1.7 – *Exemple de calcul d'une machine de Turing qui ajoute* 1 *au nombre codé en binaire sur son ruban.*

Les machines de Turing sont un modèle de calcul robuste, au sens où certaines modifications de leur définition ne modifie par leur puissance de calcul.

Proposition 1.4.3. *Les machines de Turing telles que décrites dans la Définition 1.4.1 sont équivalentes aux modèles suivants : MT sur un alphabet à deux lettres, MT avec un ruban semi-infini, MT avec k rubans et k têtes de calculs, MT avec ruban bi-dimensionnel, MT où la tête est immobile et c'est le ruban qui se déplace.*

Les preuves de ces équivalences sont laissées en (*exercice*), on pourra en trouver les détails dans [123].

Remarque 1.4.4. *Si $\mathcal{M}$ est une machine de Turing, on peut construire automate cellulaire F de voisinage $\{-1, 0, 1\}$ qui code le comportement de la machine $\mathcal{M}$ (*exercice*).*

1.4.2 Problème de l'arrêt

Le *problème de l'arrêt* consiste à déterminer, étant données une machine de Turing $\mathcal{M}$ et une entrée x, si la machine $\mathcal{M}$ atteint un état final (s'arrête) au cours de son calcul initié sur l'entrée x.

Théorème 1.4.5 (Turing [132])**.** *Le problème de l'arrêt est indécidable.*

Démonstration. La preuve utilise un argument diagonal, qu'on peut trouver dans [123] par exemple. ■

Une variante du problème de l'arrêt, appelée *problème de l'arrêt sur l'entrée vide*, consiste à déterminer si une machine de Turing $\mathcal{M}$ s'arrête au cours de son calcul initié sur l'entrée vide (ruban initialement rempli de symboles ♯).

Théorème 1.4.6. *Le problème de l'arrêt sur l'entrée vide est indécidable.*

Démonstration. On procède par réduction à partir du problème de l'arrêt (*exercice*). ■

1.4.3 MT et règles locales

Un machine de Turing n'est finalement rien de plus qu'un automate fini se déplaçant sur un ruban infini et, et c'est ce qui nous intéresse plus particulièrement ici, la machine a besoin de ne connaître qu'une quantité finie d'information pour calculer (le contenu du ruban à la position de

la tête de calcul et l'état de cette dernière). Pour cette raison, on peut aisément encoder le comportement d'une machine de Turing, c'est-à-dire son ensemble de règles, dans un SFT dont on détaille à présent les motifs autorisés. La dimension horizontale joue le rôle du ruban de la machine, tandis que la dimension verticale représente l'évolution du temps (qui s'écoule de bas en haut). On obtient ainsi les *diagrammes espace-temps* de la machine $\mathcal{M}$, qu'on peut construire à l'aide des motifs autorisés de taille 3×2 suivants :

— si le motif code une partie du ruban dans laquelle la tête de calcul n'apparaît pas, les deux lignes du motif sont identiques et pour $x, y, z \in \Gamma$ on autorise le motif :

x	y	z
x	y	z

— si la tête de calcul est présente dans la partie du ruban, on code les règles de transition de la machine. Par exemple la règle $\delta(q_1, x) = (q_2, y, \leftarrow)$ sera codée par les motifs :

v	w	(q_2, z)
v	w	z

w	(q_2, z)	y
w	z	(q_1, x)

(q_2, z)	y	z'
z	(q_1, x)	z'

y	z'	z''
(q_1, x)	z'	z''

— enfin si q_f est un état final, alors la machine arrête son calcul, ce qu'on traduit par le motif :

z	(q_f, x)	z'
z	(q_f, x)	z'

On note par $P_{\mathcal{M}}$ l'ensemble des motifs interdits sur l'alphabet $\mathcal{A}_{\mathcal{M}} = \Gamma \cup (Q \times \Gamma)$ construit à partir des règles de transition de $\mathcal{M}$, c'est-à-dire les motifs qui ne sont pas représentés dans la liste ci-dessus.

Considérons à présent le décalage de type fini $\mathbf{X}_{P_{\mathcal{M}}}$. Il contient tous les diagrammes espace-temps de la machine $\mathcal{M}$, mais aussi d'autres configurations qui ne sont jamais atteintes par la machine. Avec la machine de Turing de l'exemple 1.4.2, le SFT $\mathbf{X}_{P_{\mathcal{M}}}$ contient une configuration dans laquelle le ruban est dans l'état suivant

$\dots$	1	$\dots$	1	1	1	1	1	1	1	$(q_+, 1)$	0	$\dots$	0	$\dots$,

état du ruban qui est incohérent puisque jamais atteint dans un calcul de $\mathcal{M}$ sur une entrée finie.

Le problème vient du manque d'information sur le début d'un calcul. Il faut spécifier un point dans $\mathbb{Z}^2$ qui jouera le rôle d'origine du calcul : à cette position la tête de calcul est dans l'état initial q_0, placée au début du mot d'entrée, en dehors duquel le ruban ne contient que des symboles blanc $\sharp$. Mais comment introduire une origine dans un décalage ? Par compacité du décalage, il est impossible d'imposer qu'un symbole origine apparaisse une fois et une seule dans chaque configuration du décalage. Nous découvrirons une solution à ce problème dans la Partie 1.5.2.

1.5 Problèmes de décision

Un SFT peut être représenté par une liste finie de motifs interdits, un automate cellulaire par sa fonction locale et un sous-décalage sofique peut être représenté par un SFT et un automate cellulaire. Il est naturel de se demander si, étant donnée une telle représentation d'un de ces objets, on peut déterminer ses propriétés globales.

1.5.1 Décidabilité des sous-décalages sofiques en dimension 1 et applications

Dans cette section un automate fini $\mathbb{A} = (G, \lambda)$ est un graphe orienté (fini) $G = (V, E)$ dont les arêtes sont étiquetées par $\lambda : E \to \mathcal{A}$. À $\mathbb{A}$ on associe l'ensemble de ses parcours bi-infinis étiquetés

$$\mathbf{X}_{\mathbb{A}} = \left\{ x \in \mathcal{A}^{\mathbb{Z}} \mid \exists y \in V^{\mathbb{Z}}, \forall z \in \mathbb{Z}, (y_z, y_{z+1}) \in E \text{ et } x_z = \lambda(y_z, y_{z+1}) \right\}.$$

$\mathbf{X}_{\mathbb{A}}$ est toujours un sous-décalage sofique : aux changements d'alphabet près, c'est l'image d'un SFT (l'ensemble des parcours bi-infinis valides dans le graphe G) par un automate cellulaire (l'étiquetage λ).

Théorème 1.5.1 (Traité en détail dans [81])**.** *Lorsque la dimension est 1, on a :*

- *si $\mathbf{X}$ est un sous-décalage sofique alors $\mathbf{X} = \mathbf{X}_{\mathbb{A}}$ pour un certain automate fini $\mathbb{A}$, plus précisément : étant donnée une paire $(F, \mathbf{Y})$ où F est un AC et $\mathbf{Y}$ un SFT, on peut calculer un automate fini $\mathbb{A}$ tel que $\mathbf{X}_{\mathbb{A}} = F(\mathbf{Y})$;*
- *réciproquement, étant donné un automate fini $\mathbb{A}$, on peut calculer une paire $(F, \mathbf{X})$ où F est un AC et $\mathbf{X}$ un SFT tels que $\mathbf{X}_{\mathbb{A}} = F(\mathbf{X})$;*
- *le langage $\mathcal{L}(\mathbf{X})$ d'un sous-décalage sofique $\mathbf{X}$ est toujours rationnel ;*
- *on peut décider l'égalité de deux sofiques donnés par automate fini, ou par paire SFT/AC .*

Grâce à cette représentation par automate fini, les sous-décalages sofiques en dimension 1 sont bien compris et apportent leur lot d'algorithmes de décision.

Théorème 1.5.2 (Amoroso-Patt [3])**.** *La surjectivité et l'injectivité sont des propriétés décidables pour les automates cellulaires 1D.*

Démonstration. Tout d'abord, la surjectivité d'un AC F revient à savoir si le sous-décalage sofique $I_1 = F(\mathcal{A}^{\mathbb{Z}^d})$ est égal au sous-décalage sofique $\mathcal{A}^{\mathbb{Z}^d}$. La proposition 1.5.1 montre que c'est décidable lorsque la dimension est 1.

Concernant l'injectivité, on peut construire pour tout F le SFT $\mathbf{X} \subseteq (\mathcal{A} \times \mathcal{A})^{\mathbb{Z}^d}$ représentant les paires de configurations qui ont la même image par F :

$$\mathbf{X} = \left\{ x \in (\mathcal{A} \times \mathcal{A})^{\mathbb{Z}^d} \mid F(\pi_1(x)) = F(\pi_2(x)) \right\}$$

où π_1 et π_2 sont respectivement les projections sur la première et la deuxième composante de l'alphabet $\mathcal{A} \times \mathcal{A}$. $\mathbf{X}$ est bien un SFT car si $\mathbb{U}$ et f sont le voisinage et la fonction de transition locale de F, on peut le définir par la liste finie de motifs interdits $u \in (\mathcal{A} \times \mathcal{A})^{\mathbb{U}}$ tels que $f(\pi_1(u)) \neq f(\pi_2(u))$. Si on note $D = \{(a, a) \mid a \in \mathcal{A}\}$, l'injectivité de F est équivalente à l'égalité $\mathbf{X} = D^{\mathbb{Z}^d}$: c'est une égalité entre sous-décalage sofiques (SFT en l'occurrence) dont on peut calculer une représentation étant donné F, donc c'est décidable d'après la proposition 1.5.1 car la dimension est 1. ■

1.5.2 Indécidabilité du problème du Domino et de ses variantes

Le problème du Domino est le suivant : étant donné un jeu fini de tuiles de Wang, est-il possible de trouver un pavage valide du plan par ce jeu de tuiles ? Il est toujours possible de détecter en temps fini si un jeu de tuiles ne permet pas de paver le plan. Par exemple, on peut commencer à paver le plan en remplissant des rectangles de plus en plus grands, avec retour en arrière en cas de blocage. Ainsi, on dispose d'un premier semi-algorithme, prenant en entrée un jeu de tuiles, et qui s'arrête si le jeu de tuiles ne pave pas le plan et boucle sinon. Supposer que le jeu de tuiles en entrée est périodique (*i.e.* tous les pavages possibles sont périodiques) permet de résoudre le problème et fait le lien avec la Conjecture 1.3.1.

Proposition 1.5.3 (Wang [134])**.** *Si la Conjecture de Wang est vraie, alors le problème du Domino est décidable.*

Démonstration. Il suffit de combiner le semi-algorithme présenté plus haut avec le semi-algorithme qui recherche de manière exhaustive un motif rectangulaire permettant de paver le plan de façon périodique (il suffit de vérifier que les couleurs sur les bords opposés du motif sont les mêmes). ■

Nous avons vu dans la Partie 1.3.2 qu'il existe des jeux de tuiles apériodiques, ce qui contredit la Conjecture de Wang mais ne nous apprend pour l'instant rien sur la décidabilité du problème du Domino.

Une variante du problème du Domino consiste à se fixer une tuile qui apparaît nécessairement dans le pavage qu'on recherche. Étant donnés un jeu de tuiles fini et une tuile de ce jeu, existe-t-il un pavage valide du plan avec cette tuile placée en l'origine ? Si on sait résoudre le problème du Domino à origine fixée, alors on sait résoudre le problème du Domino : il suffit de chercher un pavage en fixant successivement comme tuile d'origine toutes les tuiles du jeu de tuiles.

Théorème 1.5.4 (Kahr, Moore & Wang [59])**.** *Le problème du Domino à origine fixée est indécidable.*

Démonstration. On combine l'encodage des règles de calcul d'une machine de Turing dans un jeu de tuiles présenté dans la Partie 1.4.3 avec le sous-décalage sofique $\mathbf{X}_{\leqslant 1}$. Ce dernier permet de forcer la présence d'une unique tête de calcul dans n'importe quel pavage, et donc assure que la machine de Turing encodée s'arrête si et seulement si le jeu de tuiles ainsi construit ne pave pas le plan. Par réduction au problème de l'arrêt sur l'entrée vide (Théorème 1.4.6), on obtient le résultat. ■

Théorème 1.5.5 (Berger [8])**.** *Le problème du Domino est indécidable.*

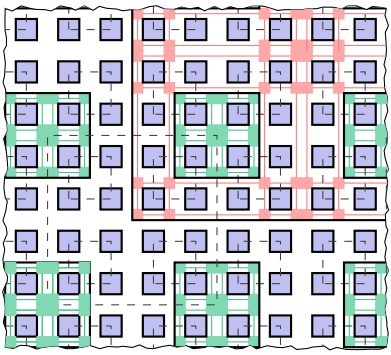


FIGURE *1.8 – Espace de calculs au sein de pavage de Robinson. En bleu, vert et rouge les zones de calcul de niveaux* 0, 1 *et* 2*.*

Démonstration. Le problème du jeu de tuiles construit dans la preuve du Théorème 1.5.4 est qu'il permet à la fois des pavages codant un seul calcul de la machine de Turing infini en espace et en temps, comme des pavages ne codant aucun calcul. Pour remédier à ce problème, on modifie le jeu

de tuiles pour faire apparaître, dans chaque pavage possible, une infinité de calculs disposant d'un espace et d'un temps fini, ces derniers pouvant cependant être arbitrairement grands. Un moyen pour y parvenir est d'utiliser la structure hiérarchique de carrés imbriqués du pavage de Robinson (voir la Figure 1.8). Pour plus de détails, voir [112]. ■

Une deuxième variante est le problème périodique du Domino : étant donné un jeu de tuiles fini, peut-on trouver un pavage périodique par ce jeu de tuiles ? Cette variante est aussi indécidable [41, 50].

Théorème 1.5.6 (Gurevich & Koryakov [41])**.** *Le problème périodique du Domino est indécidable.*

1.5.3 Pavages déterministes et nilpotence en toutes dimensions

Étant donné un jeu de tuiles de Wang T on peut définir un automate cellulaire F de dimension 2 sur l'alphabet $\mathcal{A} = T \cup \{x\}$ où $x \notin T$ et de voisinage $\mathbb{U} = \{\mathbf{i} \in \mathbb{Z}^2 \mid \|\mathbf{i}\|_\infty \leqslant 1\}$ au comportement suivant :

— si le voisinage d'une cellule contient un x ou une erreur de pavage, alors la cellule passe dans l'état x ;

— sinon la cellule est inchangée.

F est nilpotent si et seulement si T n'admet aucun pavage valide du plan (*exercice*).

Cette simple construction montre par réduction au problème du Domino que la nilpotence est indécidable pour les automates cellulaires de dimension 2 et plus. L'espace de l'automate cellulaire correspond ici à l'espace des pavages. Le cas de la dimension 1 demande plus de travail et met en lien l'espace des pavages avec l'espace-temps des automates cellulaires.

Un jeu de tuiles T est dit NE-déterministe (NE = nord est) si pour toute paire de tuile $\tau_1, \tau_2 \in T$ il y a *au plus* une tuile $\tau_{NE} \in T$ telle que le motif « τ_1 à l'ouest de τ_{NE} et τ_2 au sud de τ_{NE} »

$$\begin{array}{cc} \tau_1 & \tau_{NE} \\ & \tau_2 \end{array}$$

ne viole pas les contraintes de pavage. Si une telle tuile τ_{NE} n'existe pas, la paire (τ_1, τ_2) est dite NE-bloquante.

Théorème 1.5.7 (Kari [60])**.** *Le problème du Domino restreint aux jeux de tuile NE-déterministes est indécidable.*

L'idée clef dans l'introduction des pavages NE-déterministes est qu'ils peuvent être vus comme des automates cellulaires partiels de dimension 1 :

les configurations sont les diagonales NO-SE de pavage par T et l'image d'une configuration, si elle existe, est l'unique diagonale décalée d'une unité vers le nord qui respecte les contraintes de pavage. Pour avoir un automate cellulaire, on ajoute un état $x \notin T$ qui apparaît en cas de paire de tuiles NE-bloquante et se propage aux cellules voisines à chaque étape. On peut alors montrer qu'un tel automate est nilpotent si et seulement si le jeu de tuiles T n'admet pas de pavage valide du plan (*exercice*). Par le théorème précédent on en déduit l'indécidabilité de la nilpotence en 1D.

Corollaire 1.5.8 (Kari [60])**.** *La nilpotence des automates cellulaires est indécidable en toute dimension.*

1.5.4 Pavages finis et surjectivité en dimension 2

Étant donné un jeu de tuiles T et une tuile t_0, un pavage valide du plan par T est t_0-fini s'il ne contient qu'un nombre fini de positions où apparaît une autre tuile que t_0. Le problème de la pavabilité finie consiste à déterminer si étant donné T et $t_0 \in T$ il existe un pavage valide du plan par T qui soit t_0-fini.

Théorème 1.5.9 (Kari [61])**.** *Le problème de la pavabilité finie est indécidable.*

Le théorème 1.2.21 affirme qu'un automate cellulaire n'est pas surjectif exactement lorsqu'il existe deux configurations qui diffèrent en un nombre fini de positions et qui ont même image. On peut construire à partir de tout jeu T et tuile t_0 un AC qui admet une telle paire de configurations si et seulement si T admet un pavage valide t_0-fini. On en déduit le théorème suivant (une première preuve a été donnée dans [61] puis simplifiée dans [30]).

Théorème 1.5.10 (Kari [61, 30])**.** *Le problème de la surjectivité est indécidable pour les automates cellulaires de dimension 2 et plus.*

1.5.5 Tuiles orientées, serpents et réversibilité en dimension 2

Un jeu de tuiles *orienté* est un jeu T muni d'une orientation de chaque tuile, *i.e.* une fonction $\vec{D} : T \to \{(\pm 1, 0), (0, \pm 1)\}$ qui à une tuile associe une direction (nord, sud, est ouest). Étant donnée une configuration $c \in T^{\mathbb{Z}^2}$ (pas nécessairement un pavage valide), un *serpent* dans c est une suite (finie ou infinie) de positions $p_1, p_2, p_3, \ldots \in \mathbb{Z}^2$ telles que pour tout n :

1. $p_{n+1} = p_n + \vec{D}(c(p_n))$;
2. le motif $\mathbf{i} \in p_n + \{(\pm 1, 0), (0, \pm 1)\} \mapsto c(\mathbf{i})$ ne viole pas les contraintes de pavage de T.

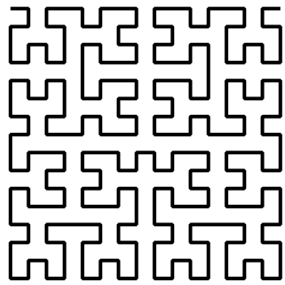


FIGURE *1.9 – 4^e itération dans la construction de la courbe de Hilbert.*

En combinant la construction de Robinson présentée en 1.3.2 et le processus itératif qui donne la courbe de Hilbert (voir figure 1.9) on peut montrer l'existence d'un jeu de tuiles orienté très particulier dont les serpents sont forcés à remplir l'espace.

Théorème 1.5.11 (Kari [61])**.** *Il existe un jeu de tuiles orienté T_0 (dit "space-filling") qui peut paver le plan et dont chaque serpent infini parcourt intégralement des carrés $N \times N$ pour des valeurs de N arbitrairement grandes.*

À partir de ce jeu de tuiles, on peut obtenir diverses constructions en ajoutant de nouvelles composantes. Par exemple, en suivant les serpents infinis on peut vérifier une propriété locale sur des carrés arbitrairement grands et ainsi tester si la composante supplémentaire admet un pavage valide du plan. On peut aussi appliquer un automate cellulaire 1D « le long » des serpents et exploiter des différences entre les serpents infinis et les serpents finis. On en déduit de nombreux résultats d'indécidabilité par réduction (directe ou indirecte) au problème du domino.

Théorème 1.5.12 (Kari [61, 63, 65])**.** *Les problèmes suivants sont indécidables :*

- *déterminer si un jeu de tuiles orienté admet un serpent infini ;*
- *déterminer si un automate cellulaire 2D est réversible ;*
- *déterminer si un automate cellulaire 2D F est périodique,* i.e. *s'il existe n tel que F^n est la fonction identité.*

Notons en particulier qu'en dimension 2 et plus aucune fonction récursive ne permet de borner la taille du voisinage de l'inverse F^{-1} d'un AC F réversible en fonction de l'alphabet et du voisinage de F. En dimension 1, au contraire, une borne optimale linéaire a été établie [26].

Enfin notons que la périodicité d'un automate cellulaire, propriété plus restrictive que la réversibilité, est elle indécidable en dimension 1 comme en dimension 2, mais la preuve connue en 1D fait appel à des techniques très différentes [66].

Chapitre 2

Une introduction aux jeux combinatoires

Eric Duchêne
Aline Parreau

Les jeux combinatoires sont des objets scientifiques qui intéressent à la fois les chercheurs en mathématiques et informatique. En termes simples, il s'agit de jeux à 2 joueurs sans hasard ni information cachée, où le vainqueur est uniquement déterminé par celui qui joue le dernier coup. Si les premières résolutions exactes de jeux datent de 1900, on considère généralement qu'il s'agit d'un domaine de recherche à part entière depuis le milieu des années 1970 et les travaux théoriques de John Conway [21]. En particulier, il a montré qu'on pouvait associer à certains jeux une valeur parmi l'ensemble des nombres surréels*. Cette valeur permet notamment d'évaluer une position de jeu qui se décompose en plusieurs sous-jeux.*

Dès lors, une communauté internationale active cherche à étoffer le cadre posé par Conway. Parmi les avancées récentes, on citera les travaux de résolution des jeux misère *et* loopy *[122] dans les années 2000. De manière générale, il est intéressant de constater que la théorie des jeux combinatoires s'est construite en empruntant des outils à de nombreux autres domaines tels que la théorie des nombre, la combinatoire des mots ou encore la théorie des graphes. Précisons enfin que l'un des défis actuels pour la communauté réside dans la compréhension théorique des jeux à score (type Go, Othello ou Dots and Boxes).*

Parallèlement à cette étude, dans le domaine de l'intelligence artificielle, les jeux combinatoires servent de support idéal pour jauger le niveau de performance des recherches. Les avancées les plus significatives y sont d'ailleurs

très souvent médiatisées. Cela a par exemple été le cas en 1999 lorsque le supercalculateur Deep Blue *d'IBM a battu le champion du monde Kasparov aux échecs. En 2007, les équipes de Jonathan Shaeffer (Université d'Alberta, Canada) ont montré après 20 ans de calculs sur plus de 50 machines qu'une partie de dames 8×8 s'achève nécessairement sur une partie nulle lorsque les deux joueurs jouent optimalement. C'est encore le cas en 2016, lorsque le champion du monde de Go, Lee Sedol, a été battu par le programme* Alpha Go *de Google qui combine des techniques de Monte Carlo avec des réseaux de neurones profonds.*

Notons également que l'étude des jeux combinatoires propose à la fois des techniques et des résultats théoriques qui peuvent se révéler être des sources pour d'autres domaines de l'informatique mathématique. On peut citer par exemple les études actuelles des théoriciens des graphes sur les paramètres d'optimisation ludiques (nombre chromatique ludique, nombre de domination ludique…). Dans ce contexte, la notion de somme de jeux combinatoires *s'adapte assez bien. Par ailleurs, les IA propices aux jeux ont également démontré leur efficacité pour d'autres problèmes de décision séquentiels comme ceux issus de la recherche opérationnelle.*

Ainsi, les jeux combinatoires forment un domaine de recherche actif aux challenges nombreux. Que cela soit en mathématiques ou en informatique, on s'intéresse la plupart du temps à leur résolution en terme de stratégie gagnante pour l'un des deux joueurs. Si le mathématicien cherchera principalement une stratégie exacte, l'informaticien s'intéressera quant à lui à la complexité algorithmique de cette recherche et, pour les jeux de haute complexité, pourra produire des algorithmes approchant au mieux la stratégie optimale. De tels algorithmes nécessitent une bonne compréhension de la théorie mathématique inhérente à ces jeux.

Ce cours a pour objectif de vous présenter la théorie qui se cache derrière les jeux combinatoires. Dans un premier temps, elle sera abordée dans un cadre général (sections 1 et 2), illustrée avec deux exemples de jeux bien connus. Dans un second temps (section 3), nous illustrerons les fortunes diverses que l'on peut rencontrer lors de l'étude de la complexité algorithmique des jeux combinatoires, au travers des familles de jeux très étudiées dans la littérature.

2.1 Introduction et premières briques

2.1.1 Jeu combinatoire : définition

Les jeux combinatoires — au sens large — sont des objets familiers pour beaucoup d'entre nous. Parmi les jeux les plus connus, on peut citer le jeu de Nim, le Go, les échecs ou encore les dames. La théorie mathématique qui permet d'envisager leur résolution est toujours en cours de construction aujourd'hui. Dans certains cas (jeux en convention normale), un cadre de résolution théorique a été construit par Conway à la fin des années 1970. C'est celui-ci que nous détaillerons dans ce cours. Dans d'autres cas (jeux avec boucles ou jeux à score par exemple), la compréhension de ces jeux s'avère un problème d'actualité pour la communauté scientifique concernée.

Une définition d'un jeu combinatoire (au sens strict du terme) a été donnée pour la première fois par Berlekamp, Conway et Guy dans leur ouvrage référence [9]. En voici les ingrédients :

— Il y a exactement deux joueurs, appelés "Left" et "Right", qui jouent à tour de rôle, sans passer leur tour ;
— l'information est totale : tous les coups possibles sont connus par les deux joueurs ;
— le hasard n'intervient pas dans les règles du jeu (sous forme de lancers de dés ou de tirages de cartes par exemple) ;
— les règles sont définies de sorte que le jeu ait toujours une fin, et le nombre de positions de jeu est finie ;
— le vainqueur est désigné par le dernier coup joué : en *convention normale*, le premier joueur qui ne peut plus jouer a perdu. C'est l'inverse en *convention misère*.

Le dernier point implique qu'il y a toujours un gagnant dans un jeu combinatoire. Nous verrons un peu plus loin qu'une définition plus formelle d'un jeu peut-être donnée en tant que jeu de déplacement sur un arbre.

Si l'on se réfère aux éléments ci-dessus, les conditions requises sont alors trop fortes pour que les jeux abstraits à deux joueurs les plus célèbres soient combinatoires. Par exemple, les échecs et les dames peuvent avoir des parties nulles. Cela provient du fait que dans ces jeux, certaines positions peuvent être visitées plusieurs fois au cours de la partie. De tels jeux sont appelés *loopy*. Au Go, à Dots and Boxes ou à Othello, le vainqueur est déterminé par un score et non en fonction du dernier coup joué. Cependant, tous ces jeux restent proches des jeux combinatoires. On peut trouver dans la littérature des éléments de réponse pour les résoudre (voir [122],

chap. 6 pour les jeux loopy et [76] pour les jeux à score). Si certains des concepts et problématiques présentés dans ce cours restent pertinents dans ces contextes étendus, nous ne développerons toutefois pas les résultats qui leur sont spécifiques. De même, la théorie des jeux en convention misère ne sera pas abordée ici (très différente de celle des jeux en convention normale).

Pour le lecteur qui souhaite accéder à un état de l'art assez complet sur le domaine, les trois livres suivants remportent les suffrages de la communauté : *Winnning Ways* [9], *Lessons in Play* [1] et *Combinatorial Game Theory* [122]. Le premier cité est l'œuvre pionnière qui fait suite aux travaux de Conway. Constitué de quatre volumes, il contient un vaste panel de jeux avec résolution à l'appui. Le deuxième est un ouvrage très clair et pédagogique, utile notamment pour faire découvrir la théorie de façon ludique et bien organisée. Le dernier livre, plus récent et plus formel, couvre avec un spectre très large les grands résultats du domaine.

2.1.2 Deux exemples fil rouge : NIM et DOMINEERING

Il est temps désormais de donner deux exemples de "vrais" jeux combinatoires qui serviront de fil rouge à ce cours. Il s'agit du jeu de NIM [13] et de DOMINEERING [37]. Pour le premier, les positions de jeu sont des n-uplets d'entiers positifs ou nuls qu'on notera $(a_1, \dots, a_n)$. Un coup consiste à soustraire à l'une des composantes a_i (avec $a_i > 0$) une valeur comprise entre 1 et a_i. Le joueur qui ne peut plus trouver un tel a_i perd la partie. En d'autres termes, celui qui joue vers la position $(0, \dots, 0)$ est le vainqueur.

Le jeu DOMINEERING se joue quant à lui sur une grille rectangulaire. À tour de rôle, les deux joueurs placent un domino sur la grille selon les contraintes suivantes : Left ne peut placer ses dominos que verticalement, et Right qu'horizontalement. Ici encore nous jouons en convention normale et le premier joueur qui ne peut plus placer de domino perd la partie. La figure 2.1 illustre une position de jeu où Left a commencé et gagne, puisque Right ne peut plus placer aucun domino horizontal sur la grille.

2.1.3 Modélisation d'un jeu

Une première propriété très utile se dégage de la définition d'un jeu : tout jeu combinatoire peut se jouer sur un certain arbre fini décrit ci-dessous.

Définition 2.1.1 (Arbre de jeu)**.** *Étant donné un jeu $\mathcal{G}$ ayant comme position de départ S, l'*arbre de jeu *associé à $\mathcal{G}$ est un arbre semi-ordonné (c'est-à-dire avec deux types de fils — gauches et droits) défini de la façon suivante :*

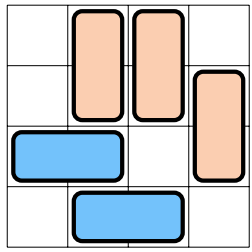


FIGURE 2.1 – *Une partie de* DOMINEERING *: Right est bloqué et perd.*

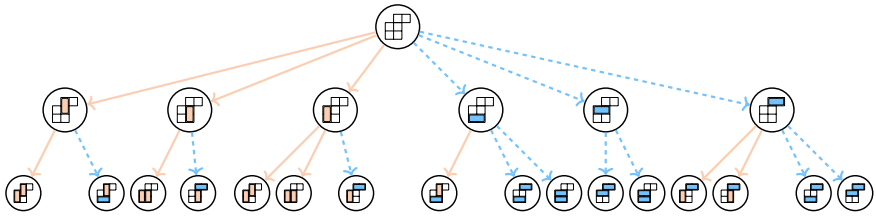


FIGURE 2.2 – *Arbre de jeu d'une position de* DOMINEERING

— *On construit un sommet racine qui correspond à la position de départ S ;*
— *on construit les fils de ce sommet de la façon suivante : toutes les positions de jeu atteignables par Left (resp. Right) en un coup sont les fils gauches (resp. droits) de la racine ;*
— *on applique la règle précédente récursivement pour tous les fils nouvellement créés.*

La figure 2.2 donne l'arbre de jeu de DOMINEERING avec comme position de départ  . On notera que seuls les trois premiers niveaux de l'arbre ont été dessinés (l'arbre complet admet un niveau supplémentaire).

Jouer à un jeu sur son arbre consiste à déplacer à tour de rôle un jeton situé initialement à la racine. Chaque joueur le déplace le long d'un arc qui lui correspond (arcs orientés vers la gauche pour Left, et vers la droite pour Right). En convention normale, le premier joueur qui ne peut plus déplacer le jeton perd la partie.

Enfin, définissons maintenant la notion d'*option* qui correspond aux fils d'un sommet de l'arbre.

Définition 2.1.2 (Option)**.** *Pour une position de jeu donnée G, on appellera* options *de G l'ensemble des positions de jeu atteignables en un coup. Celles-ci peuvent être partitionnées en deux sous-ensembles : les options gauches (positions atteignables par Left) et les options droites (positions atteignables par Right).*

2.1.4 Problématiques en théorie des jeux combinatoires

Étant donné un jeu, les chercheurs en théorie des jeux combinatoires s'intéressent à trois problématiques majeures :

— Si l'on suppose que les deux joueurs jouent au mieux à chaque tour, qui sera le vainqueur ?
— quelle est la valeur du jeu ? Cette notion sera définie dans la partie 2.2 et constitue le cœur de la théorie ;
— peut-on construire une stratégie gagnante pour le vainqueur ? (c'est-à-dire une séquence de coups qui amène à la victoire, quels que soient les coups de son adversaire).

Grâce à l'arbre de jeu (dont la taille est finie), il n'est pas difficile de voir que ces problèmes sont tous décidables. C'est donc la complexité algorithmique de leur résolution qui va nous intéresser. Avant de rentrer dans plus de formalisme, abordons-les à travers quelques exemples.

Prenons le jeu de NIM avec $n = 2$. On peut remarquer que les positions de jeu de la forme (a_1, a_2) avec $a_1 \neq a_2$ sont gagnantes pour le joueur qui commence. La stratégie gagnante consiste à rendre les deux valeurs égales. L'autre joueur n'aura pas d'autre choix que de les rendre différentes, et ainsi de suite jusqu'à la position finale $(0, 0)$. Lorsque $n > 2$, nous verrons plus tard que les problèmes restent solubles en temps polynomial, bien que la résolution soit moins immédiate.

Il existe d'autres jeux pour lesquels on sait dire qui gagne, mais sans pour autant connaître une stratégie gagnante en temps raisonnable : c'est le cas par exemple de CHOMP ou de HEX. Dans le cas de DOMINEERING sur un grille $n \times m$, la complexité de ces trois problèmes est ouverte.

Nous donnons par la suite quelques éléments supplémentaires à propos des problèmes décrits précédemment.

2.1.5 Recherche de l'issue du jeu

L'*issue* d'un jeu combinatoire correspond au problème de la détermination du vainqueur. Celle-ci ne peut prendre que quatre valeurs possibles :

— $\mathcal{L}$ si Left admet une stratégie gagnante indépendamment de qui commence ;
— $\mathcal{R}$ si Right admet une stratégie gagnante indépendamment de qui commence ;
— $\mathcal{N}$ si le premier joueur admet une stratégie gagnante ;
— $\mathcal{P}$ si le deuxième joueur admet une stratégie gagnante.

Étant donné un jeu G, on notera par la suite $o(G)$ l'issue de G. Le fait que seulement quatre issues soient possibles se déduit directement de l'arbre

de jeu en étiquetant les sommets des feuilles jusqu'à la racine. Toutes les feuilles sont initialement étiquetées $\mathcal{P}$ car celui qui doit jouer depuis une feuille est bloqué et donc déclaré perdant (en convention normale uniquement). Puis on étiquette récursivement les sommets des niveaux précédents en utilisant la proposition suivante :

Proposition 2.1.3. *Un nœud d'un arbre de jeu sera étiqueté :*
- *$\mathcal{P}$ si ses fils gauches sont tous $\mathcal{N}$ ou $\mathcal{R}$ et ses fils droits sont tous $\mathcal{N}$ ou $\mathcal{L}$;*
- *$\mathcal{N}$ si au moins un fils gauche est $\mathcal{L}$ ou $\mathcal{P}$ et au moins un fils droit est $\mathcal{R}$ ou $\mathcal{P}$;*
- *$\mathcal{L}$ si au moins un fils gauche est $\mathcal{L}$ ou $\mathcal{P}$ et ses fils droits sont tous $\mathcal{N}$ ou $\mathcal{L}$;*
- *$\mathcal{R}$ si ses fils gauches sont tous $\mathcal{N}$ ou $\mathcal{R}$ et au moins un fils droit est $\mathcal{R}$ ou $\mathcal{P}$.*

Cet algorithme permet donc de calculer l'issue d'un jeu en temps polynomial en fonction de la taille de l'arbre. Cependant, la taille d'une position de jeu est souvent bien plus petite que la taille de l'arbre correspondant. Par exemple, une position $(a_1, \dots, a_n)$ du jeu de NIM (avec $a_i > 0$ pour tout i) admet un codage de taille $\mathcal{O}(\sum_{i=1}^{n} \log_2(a_i))$, ce qui est bien plus faible que le nombre de sommets dans l'arbre de jeu (de l'ordre de $\prod_{i=1}^{n} a_i$). Par conséquent, la construction de l'arbre de jeu n'est souvent pas la solution pour calculer en temps polynomial l'issue d'un jeu. D'autres caractérisations doivent alors être envisagées. Dans le cas de NIM, il a été prouvé [13] qu'une position $(a_1, \dots, a_n)$ est $\mathcal{P}$ si et seulement si $a_1 \oplus \cdots \oplus a_n = 0$, où $\oplus$ est l'opérateur d'addition bit à bit sans retenue (XOR), qu'on appellera *Nim-somme*. Toutes les autres positions sont $\mathcal{N}$. Donnons quelques éléments de preuve de ce résultat :

Théorème 2.1.4. *Une position $(a_1, \dots, a_n)$ de* NIM *est $\mathcal{P}$ si et seulement si $a_1 \oplus \cdots \oplus a_n = 0$.*

Preuve. La seule position finale du jeu est la position nulle $(0, \dots, 0)$, qui est trivialement $\mathcal{P}$, et vérifie bien la propriété de Nim-somme nulle. Considérons maintenant une position non nulle quelconque $(a_1, \dots, a_n)$ telle que $a_1 \oplus \cdots \oplus a_n = 0$. D'après la proposition 2.1.3, montrons que tout coup depuis cette position produit une position où la Nim-somme est non-nulle. Le joueur qui joue à partir de cette position donne une position $(a'_1, \dots, a'_n)$, où tous les a'_i sont égaux aux a_i sauf exactement pour une valeur a'_i telle que $a'_i < a_i$. Par conséquent, cette position vérifie bien $a'_1 \oplus \cdots \oplus a'_n \neq 0$.

Il nous reste alors à montrer que pour une position $(a_1, \dots, a_n)$ de Nim-somme non nulle, il existe un coup vers une position de Nim-somme nulle. Soit $S = a_1 \oplus \cdots \oplus a_n \neq 0$, et notons $\{i_1, \dots, i_k\}$ l'ensemble des indices des bits de S égaux à 1. Cet ensemble est non vide, et sans restreindre

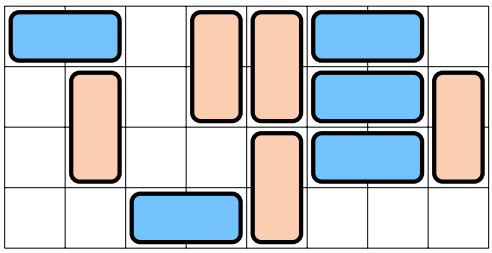


FIGURE 2.3 – *Somme de positions de* DOMINEERING

la généralité, supposons que i_1 est l'indice du bit de poids le plus fort. Par définition de S, il existe nécessairement une composante a_i dont le bit d'indice i_1 vaut 1. Il suffit alors de jouer dans cette composante a_i vers un certain a'_i obtenu en mettant tous les bits de a_i d'indices $\{i_1, \ldots, i_k\}$ à leur opposé. On a bien $a'_i < a_i$ et $a_1 \oplus \cdots \oplus a_{i-1} \oplus a'_i \oplus a_{i+1} \cdots \oplus a_n = 0$. ■

Ainsi, il n'existe aucune position $\mathcal{L}$ ou $\mathcal{R}$ au jeu de NIM. C'est d'ailleurs le cas pour tous les jeux dits *impartiaux*. Ce sont des jeux dont l'arbre est symétrique : tout coup disponible pour Left est aussi disponible pour Right, et réciproquement. Les jeux qui ne satisfont pas cette propriété (comme DOMINEERING) sont dits *partisans*.

Pour résumer la problématique de l'issue, on cherche à déterminer la complexité algorithmique pour décider si un jeu donné G est $\mathcal{P}$, $\mathcal{R}$, $\mathcal{L}$, ou $\mathcal{N}$. Dans le cas de DOMINEERING, on a vu précédemment que cette complexité n'était toujours pas connue. On connaît toutefois les issues pour certaines instances particulières (grilles avec un côté pair, grilles $5 \times n$, ou grilles carrées jusqu'au 10×10).

2.1.6 Somme de jeux

Si le calcul de l'issue est souvent un problème de haute complexité (PSPACE-difficile pour beaucoup de jeux, comme nous le verrons dans le chapitre 2.3), cette complexité peut parfois être moins élevée pour les jeux dont les positions peuvent se décomposer en une union de sous-jeux indépendants. Prenons par exemple la position de DOMINEERING de la figure 2.3.

Cette position peut être vue comme l'union (on parlera plutôt de *somme*) des trois composantes  ,  , et  . Jouer à une somme de deux jeux G_1 et G_2 revient alors à jouer à chaque tour soit sur G_1, soit sur G_2. La partie s'arrête quand les deux jeux sont terminés. Notons que cette

définition a du sens même si les deux jeux ont des règles différentes. Par exemple, on peut considérer la somme $G_1 + G_2$ où G_1 est une position de NIM $(a_1, \ldots, a_n)$ et G_2 une position de DOMINEERING.

L'intérêt principal d'une décomposition en somme de sous-jeux est la détermination de l'issue sachant l'issue de chacun des sous-jeux. On peut par exemple assez facilement prouver la propriété suivante :

Proposition 2.1.5. *Étant donné un jeu G tel que $o(G) = \mathcal{P}$, alors pour tout jeu H, on a $o(G + H) = o(H)$.*

Preuve. Sans restreindre la généralité, supposons que Left a une stratégie gagnante sur H. Alors sur $G + H$, Left applique exactement sa stratégie gagnante sur H, et à chaque fois que Right joue sur G, il répond aussi sur G en appliquant sa stratégie gagnante en tant que second joueur sur G. Ainsi, Left est certain de jouer le dernier coup à la fois sur G et H, et donc sur $G + H$. ■

Autrement dit, toutes les composantes $\mathcal{P}$ d'une somme de jeu n'influent pas sur l'issue globale du jeu. Malheureusement, la connaissance des issues des termes d'une somme de jeux est souvent insuffisante. Le tableau 2.1 résume ce qu'on peut conclure :

+	$\mathcal{P}$	$\mathcal{N}$	$\mathcal{L}$	$\mathcal{R}$
$\mathcal{P}$	$\mathcal{P}$	$\mathcal{N}$	$\mathcal{L}$	$\mathcal{R}$
$\mathcal{N}$	$\mathcal{N}$	?	$\mathcal{N}$ ou $\mathcal{L}$	$\mathcal{N}$ ou $\mathcal{R}$
$\mathcal{L}$	$\mathcal{L}$	$\mathcal{L}$ ou $\mathcal{N}$	$\mathcal{L}$	?
$\mathcal{R}$	$\mathcal{R}$	$\mathcal{R}$ ou $\mathcal{N}$	?	$\mathcal{R}$

TABLE 2.1 – *Issues des sommes de jeux*

Ainsi, la somme de deux positions $\mathcal{L}$ et $\mathcal{R}$ peut être $\mathcal{P}$, $\mathcal{N}$, $\mathcal{L}$ ou $\mathcal{R}$ selon les cas. Dans de telles situations (comme celle de la figure 2.3 qui est une somme de type $\mathcal{L} + \mathcal{L} + \mathcal{R}$), la seule notion d'issue ne suffit pas et la caractérisation doit être affinée. Conway [21] a alors réussi à généraliser la notion d'issue en attribuant une valeur à chaque jeu. Cette valeur permettra notamment de mieux appréhender le traitement des sommes. Nous allons voir dans le chapitre suivant comment ces valeurs sont construites.

2.2 Les valeurs d'un jeu combinatoire

Nous donnons dans ce chapitre un aperçu de la théorie de Conway qui associe une valeur à tout jeu combinatoire. La plupart des résultats énoncés ci-dessous sont accessibles (avec leurs preuves) dans [122].

2.2.1 Relation d'équivalence entre jeux

Dans ce qui suit, on assimilera un jeu à une position particulière d'un jeu combinatoire. NIM et DOMINEERING seront quant à eux appelés des ensembles de règles. Par conséquent, une position $(1, 2, 5)$ de NIM est un jeu. On passe d'un jeu à un autre en jouant un coup. Le fait de considérer les positions de jeu comme les éléments atomiques de la théorie permettra notamment d'accéder à la clôture de l'ensemble des jeux pour la somme.

Comme nous l'avons vu précédemment, un jeu peut être représenté par un arbre dont il sera la racine. Ainsi, on peut le définir récursivement par la seule donnée de ses fils. Étant donné un jeu G, si l'on note $G_1^L, G_2^L, \ldots, G_m^L$ (resp. $G_1^R, G_2^R, \ldots, G_n^R$) les options gauches (resp. droites) de G, alors on écrira

$$G = \{G_1^L, G_2^L, \ldots, G_m^L | G_1^R, G_2^R, \ldots, G_n^R\}.$$

Pour simplifier, on écrira plus légèrement (avec un petit abus de notation) $G = \{G^L|G^R\}$ où G^L et G^R dénotent respectivement les ensembles d'options gauches et droites de G.

Le cas de base de cette construction correspond aux arbres avec un seul nœud, autrement dit au jeu où personne ne peut jouer. On appellera ce jeu le jeu 0, il sera noté $\{\ |\ \}$. Par la suite, on définira l'ensemble des jeux combinatoires comme l'union infinie des ensemble de jeux dont l'arbre est de hauteur n, pour $n \geqslant 0$. Ainsi, on posera :

$$\mathbb{G}_0 = \{0\}$$

et récursivement les jeux de *hauteur n* seront définis comme :

$$\mathbb{G}_{n+1} = \{\{G^L|G^R\} \mid G^L, G^R \subset \mathbb{G}_n\}.$$

L'ensemble des jeux combinatoires sera alors

$$\bigcup_{n \geqslant 0} \mathbb{G}_n.$$

Avec cette notation, la somme de deux jeux G et H se définit inductivement comme suit :

$$G + H = \{G^L + H, G + H^L | G^R + H, G + H^R\},$$

avec, comme cas de base, $G + \emptyset = G$.

Définition 2.2.1 (Négation)**.** *La négation d'un jeu G, notée $-G$, est le jeu où les rôles de Left et Right sont inversés. On a $-G = \{-G^R | -G^L\}$ et cette définition est récursive.*

Avec cette notation, on pourra écrire $G - H$ pour $G + (-H)$. De plus, le jeu $-(-G)$ est isomorphe à G (leurs arbres respectifs sont les mêmes), et $o(G - G) = \mathcal{P}$.

Venons-en maintenant à la relation fondamentale des jeux en convention normale :

Définition 2.2.2 (Équivalence)**.** *Étant donnés deux jeux G et H, on définit la relation $=$ suivante entre G et H :*

$$G = H \text{ si pour tout jeu } X, o(G + X) = o(H + X).$$

Cette relation est clairement une relation d'équivalence. On notera toutefois que deux jeux isomorphes (ayant le même arbre) sont forcément égaux, mais que la réciproque n'est pas toujours vraie (on peut trouver deux jeux avec des arbres différents mais qui satisfont la relation ci-dessus).

Cette relation a permis à Conway de définir la valeur d'un jeu :

Définition 2.2.3 (Valeur d'un jeu)**.** *Étant donné un jeu G, la* valeur *de G, notée $v(G)$, est la classe d'équivalence de G modulo la relation $=$.*

On notera $\mathbb{G}$ l'ensemble des valeurs de tous les jeux possibles.

L'un des premiers résultats faciles qui découle de cette définition est le fait que dans une somme de jeux, on peut toujours remplacer un terme par un autre jeu qui lui est équivalent :

Proposition 2.2.4. *Étant donné un jeu G et deux jeux H et H' tels que $H = H'$, on a $G + H = G + H'$.*

Notons que la relation d'équivalence $=$ peut tout-à-fait être étendue dans d'autres contextes que les jeux combinatoires en convention normale, en faisant varier les notions d'issue, de somme ou de "pour tout jeu". Dans de tels contextes (jeux misère, jeux à score par exemple), certaines des propriétés que nous allons découvrir maintenant ne sont cependant plus vraies.

2.2.2 $\mathbb{G}$ est un groupe

L'une des propriétés les plus remarquables est que l'ensemble $\mathbb{G}$ muni de l'opérateur $+$ forme un groupe abélien. Examinons rapidement pourquoi.

Commençons par une propriété que nous avons déjà rencontrée, celle de l'élément neutre. Par définition du jeu noté 0, on a clairement que pour tout jeu G, $G + 0 = G$ (puisqu'aucun coup n'est jouable depuis 0).

Une question naturelle est la détermination de la classe d'équivalence de 0. La réponse est donnée par le théorème suivant :

Théorème 2.2.5. *$G = 0$ si et seulement si $o(G) = \mathcal{P}$.*

Preuve. Directement déduite de la proposition 2.1.5. ■

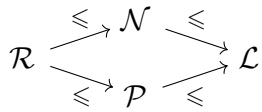
Ce résultat implique que les jeux $\mathcal{P}$ peuvent être retirés d'une somme de jeux sans en changer la valeur.

Venons-en à la question de l'inverse. Pour tout jeu G, il existe un jeu H tel que $G + H = 0$. Le candidat naturel pour H est le jeu négatif de G, en jouant symétriquement :

Proposition 2.2.6. *Pour tout jeu G, on a $G - G = 0$.*

Preuve. Issue du théorème 2.2.5 avec $o(G - G) = \mathcal{P}$. ■

Comme l'opérateur $+$ est associatif et commutatif, on a bien une structure de groupe abélien. On peut ajouter que ce groupe peut être muni d'une relation d'ordre partiel basée sur une priorité donnée à Left. En définissant en effet l'ordre partiel suivant sur les issues :



on peut définir l'ordre partiel suivant sur les jeux :

Définition 2.2.7. *Étant donné deux jeux G et H, on a*

$$G \geqslant H \text{ si } o(G + X) \geqslant o(H + X) \text{ pour tout jeu } X.$$

Cette définition permet notamment de décider de l'issue d'un jeu en fonction de sa position vis à vis du 0. En effet, les jeux $\mathcal{L}$ vérifient $G > 0$, les jeux $\mathcal{R}$ vérifient $G < 0$, les jeux $\mathcal{P}$ vérifient $G = 0$ (on le savait déjà !), et les jeux $\mathcal{N}$ sont incomparables avec 0.

2.2.3 Forme canonique d'un jeu

Si la définition récursive de la valeur d'un jeu semble rendre difficile toute simplification, il existe toutefois deux règles générales qui permettent de simplifier l'arbre tout en conservant la valeur du jeu.

Dans ce qui suit on considère un jeu G :

$$G = \{G_1^L, G_2^L, \ldots, G_m^L | G_1^R, G_2^R, \ldots, G_n^R\}.$$

Proposition 2.2.8 (Domination)**.** *Supposons que parmi les options gauches de G, il existe une position inférieure ou égale à une autre. Sans restreindre la généralité, supposons $G_1^L \leqslant G_2^L$. Alors cette position est dite* dominée *et le jeu $H = \{G_2^L, \ldots, G_m^L | G_1^R, G_2^R, \ldots, G_n^R\}$ est équivalent à G.*

Moralement, le coup G_1^L étant moins favorable à Left que G_2^L, Left ne le choisira jamais et le sous-arbre qui lui correspond n'est pas utile. Notons qu'on a le symétrique de cette proposition pour Right (si $G_1^R \geqslant G_2^R$, on peut enlever G_1^R sans affecter la valeur du jeu).

La seconde propriété de simplification est moins évidente :

Proposition 2.2.9 (Réversibilité)**.** *Supposons que pour une option gauche de G, disons G_1^L, il existe une réponse de Right vers un jeu noté G_1^{LR}, de valeur inférieure ou égale à G. Alors la position G_1^L est dite réversible, et le jeu $H = \{G_1^{LR^L}, G_2^L, \ldots, G_m^L | G_1^R, G_2^R, \ldots, G_n^R\}$ est équivalent à G, où $G_1^{LR^L}$ est l'ensemble des jeux atteignables pour Left depuis G_1^{LR}.*

Preuve. La preuve formelle de ce résultat est accessible dans [1] (chapitre 4) ou [122] (p. 65). La signification de la réversibilité peut toutefois s'expliquer assez bien : comme $G_1^{LR} \leqslant G$, on peut considérer que si Left choisit de jouer G_1^L, alors Right décide de répondre automatiquement G_1^{LR} car la position obtenue est plus favorable pour lui que la position de départ. Dans ce cas, Left a conscience que s'il choisit de jouer G_1^L, cela revient à choisir parmi les options gauches de G_1^{LR}, d'où la substitution définie dans la proposition. ■

Ainsi, un arbre de jeu peut être réduit en supprimant tous les sommets correspondant à des positions dominées ou en substituant des sommets correspondant à des positions réversibles par d'autres jeux plus simples. On considérera qu'un jeu est sous sa *forme canonique* s'il n'existe plus aucun sommet de l'arbre qui soit dominé ou réversible. Autrement dit, la forme canonique est atteinte lorsqu'on a appliqué toutes les simplifications possibles de l'arbre de jeu. On peut alors montrer que tout jeu admet une et une seule forme canonique qui lui soit équivalente (preuve dans [122], Théorème 2.9).

2.2.4 Simplifier les valeurs d'un jeu

Rappelons que la problématique initiale qui a conduit à l'introduction des valeurs d'un jeu est celle du calcul de l'issue pour les sommes de jeux. De façon plus générale, on serait encore plus heureux de pouvoir évaluer la valeur de $G + H$ connaissant celles de G et H. Existe-t-il des jeux pour lesquels on aurait $v(G + H) = v(G) + v(H)$ en interprétant l'addition comme l'addition classique de nombres ? Cela nécessiterait bien entendu l'existence d'une application allant des classes d'équivalence de $=$ vers certains nombres.

Certains jeux sont des nombres

Dans certains cas, un plongement des valeurs de jeu vers certains nombres va permettre de répondre favorablement à cette question. Commençons par définir les jeux entiers positifs :

Définition 2.2.10 (Jeux entiers positifs)**.** *Le jeu* 1 *est défini comme le jeu de valeur* $\{0|\ \}$*. Pour tout entier* $i > 1$*, le jeu* i *est défini comme étant le jeu* $(i-1)+1$*.*

Cette définition traduit assez naturellement que les jeux i avec $i > 0$ sont des jeux où Left possède i coups d'avance sur Right avant d'être bloqué. En d'autres termes, on peut aussi écrire $i = \{i-1|\ \}$. Par exemple, dans DOMINEERING, la position  vaut 2 car Left peut jouer deux fois avant d'être bloqué (et Right est bloqué immédiatement).

On définit les jeux entiers négatifs de la même façon, le jeu $-i$ étant la négation du jeu i et peut s'écrire $-i = \{\ |-i+1\}$.

Pour ces jeux-là, on a bien $v(G + H) = v(G) + v(H)$. Mais peut-on avoir d'autres nombres que les entiers ? Par exemple, quelle serait la valeur du jeu $G = \{0|1\}$? Clairement, on peut montrer que $0 < G < 1$. On est alors tenté de lui donner la valeur $1/2$. Pour vérifier que cette valeur serait bien consistante avec l'addition, il faut démontrer que $G + G = 1$. On laissera le lecteur démontrer ce résultat et on considérera désormais :

Définition 2.2.11 (Jeu $1/2$)**.** *Le jeu* $1/2$ *est défini comme le jeu de valeur* $\{0|1\}$*.*

On peut constater que le jeu  vaut précisément $1/2$.

On peut ensuite récursivement définir tous les autres jeux de la forme $\frac{1}{2^m}$ comme étant les jeux de valeur $\{0|\frac{1}{2^{m-1}}\}$. Et de manière plus générale, en sommant de tels jeux, on peut définir tous les jeux de la forme $\frac{m}{2^n}$. Ce sont les nombres dyadiques.

Ainsi, l'ensemble des jeux qui ont pour valeur des nombres dyadiques forment un sous-groupe de $\mathbb{G}$. Plus généralement, on peut montrer que pour un jeu G dont toutes les valeurs de G^L sont des nombres dyadiques inférieurs à toutes les valeurs de G^R, sa valeur est elle-même un nombre dyadique. Ainsi, le jeu $\{1/2, 2, -2|3, 7\}$ sera un nombre dyadique. De plus, il est facile de calculer la valeur du jeu en connaissant celles de ses options via la règle de simplicité :

Proposition 2.2.12 (Règle de simplicité)**.** *Étant donné $G = \{G^L|G^R\}$ où G^L et G^R sont des nombres dyadiques tels que $G^L < G^R$, le jeu G a pour valeur le nombre dyadique le plus simple compris entre G^L et G^R (c'est-à-dire celui ayant le plus petit dénominateur, et à dénominateur égal, celui ayant le plus petit numérateur en valeur absolue).*

Preuve. Soit x le nombre dyadique le plus simple entre G^L et G^R. On montre que $G - x = 0$ en montrant successivement $G - x \geqslant 0$ et $G - x \leqslant 0$ (les preuves pour chaque inégalité sont similaires). Prouver que $G - x \geqslant 0$ revient à montrer que si Right commence, il perd. S'il joue vers $G^R - x$, il perd car cette position a une valeur positive. S'il joue vers $G - x^L$, alors la simplicité de x garantit le fait que x^L n'est pas compris entre G^L et G^R (en effet, la simplicité d'un nombre se traduit par la hauteur de son arbre, et x^L a un arbre moins haut que x comme il s'agit d'une de ses options). Plus précisément, on a nécessairement $x^L \leqslant G^L$, et ainsi, Left peut répondre vers $G^L - x^L$ en s'assurant la victoire. ■

Par exemple, le jeu $G = \{1|7\}$ a pour valeur 2.

Cette règle combinée à la règle de domination permet notamment de calculer la valeur des jeux dont toutes les options gauches ont des valeurs inférieures à toutes les options droites.

Ainsi, la somme de deux jeux qui sont des nombres dyadiques est aussi un nombre dyadique obtenu simplement en faisant l'addition usuelle de ces deux nombres. Par exemple, sur la figure 2.3, la somme vaut $1/2 + 1/2 - 1 = 0$ et donc la position globale est $\mathcal{P}$.

Remarque 2.2.13. *Dans sa théorie, Conway définit les nombres comme les jeux vérifiant la propriété $x < y$ pour toute option $x \in G^L$ et $y \in G^R$. Moralement, cela correspond aux jeux où personne n'a vraiment intérêt à jouer car pour les deux joueurs, les options ont des valeurs moins intéressantes que celle de la position initiale. Les nombres dyadiques vérifient bien cette propriété. Par ailleurs, la règle de simplicité garantit que les seuls nombres de la théorie de Conway (selon cette définition) sont les nombres dyadiques. Par ailleurs, Conway a également considéré les jeux ayant des options infinies (ce que nous ne présentons pas dans*

ce manuscrit), et c'est ainsi qu'il a pu définir tous les autres nombres — réels et même surréels — comme valeurs de tels jeux.

Tous les jeux ne sont pas des nombres dyadiques

Certains jeux comme HACKENBUSH [9] ont la particularité que toutes leurs positions ont pour valeur des nombres dyadiques. Si les nombres dyadiques correspondent à des jeux faciles à traiter en termes de somme, tous les jeux ne correspondent pas à des nombres dyadiques. Par exemple, le jeu $\{0|0\}$ ne sera pas considéré dans la théorie de Conway comme un nombre dyadique, ni même comme un nombre réel ou surréel. Plus précisément, on peut montrer que $\{0|0\} < x$ pour tout jeu $x > 0$ qui est un nombre dyadique, et $\{0|0\} > y$ pour tout jeu $y < 0$ qui est un nombre dyadique. De tels jeux sont appelés des *infinitésimaux* :

Définition 2.2.14 (Infinitésimal)**.** *Un jeu G est* infinitésimal *si $x > G > -x$ pour tout nombre positif dyadique x.*

Théorème 2.2.15 (Théorème de Lawnmower [9])**.** *Si un jeu G a son arbre qui vérifie que pour tout coup de Left il existe un coup pour Right (et vice versa), alors ce jeu admet une valeur infinitésimale.*

Preuve. Pour tout $x > 0$, on montre que Left gagne sur $x - G$ quel que soit le premier joueur. La preuve se fait en considérant les deux cas (Left commence et Right commence), et en utilisant le fait que par induction, les options de G^L et G^R vérifient aussi le théorème. ■

Moralement, ce résultat signifie qu'aucun des deux joueurs n'aura de coup d'avance sur l'adversaire et donc que la valeur du jeu est proche de 0. C'est en particulier le cas pour les jeux impartiaux.

Voyons quelques exemples de jeux infinitésimaux. Prenons les positions de NIM avec $n = 1$. En notant $*i$ la valeur du jeu (i), on a :

$$*0 = \{\ |\ \}, \quad *1 = \{0|0\}, \quad *2 = \{*0, *1|*0, *1\},$$
$$*3 = \{*0, *1, *2|*0, *1, *2\}, \quad \ldots$$

Ces valeurs sont appelées des *nimbers*. Tous les jeux impartiaux sont équivalents à un nimber et donc à un jeu de NIM à une seule composante. On notera au passage que $*0 = 0$, et on simplifiera plus souvent $*1$ en $*$. De plus, la valeur d'un jeu impartial dont les options sont des nimbers est connu :

Théorème 2.2.16. *Étant donné $G = \{G^L|G^R\}$ où $G^L = G^R$ est un ensemble de nimbers, G a pour valeur le nimber $*c$, où $*c$ est le plus petit nimber non présent dans G^L.*

Preuve. Il suffit de montrer que $G + *c = 0$, autrement dit que $G + *c$ est perdant pour le premier joueur. Si celui-ci choisit une option de G^L, alors il laisse à son adversaire un jeu $*a + *c$ avec $a \neq c$. Il s'agit là d'un jeu de Nim en dimension 2 qui est gagnant pour le premier joueur d'après le théorème 2.1.4. Si en revanche le premier joueur joue dans la composante $*c$, il laisse par définition une somme $G + *b$ avec $b < c$ à son adversaire. Par minimalité de c, le nimber $*b$ appartient à G^L et G^R. Le second joueur choisit alors de répondre dans G en jouant vers le jeu $*b + *b$, qui vaut 0. Le premier joueur perdra donc dans tous les cas. ■

Par exemple, on a $\{*0, *1, *4| * 0, *1, *4\} = *2$. À partir d'un arbre de jeu impartial, ce théorème permet de construire un algorithme qui calcule la valeur de chaque sommet de l'arbre avec un algorithme similaire à celui décrit en section 1.5.

Enfin, jouer sur une somme de jeux dont les valeurs sont des nimbers est facile à appréhender depuis les travaux de Sprague et Grundy [124] :

Théorème 2.2.17. *Étant donné deux jeux de valeurs $*a$ et $*b$, on a $*a + *b = *c$ où $c = a \oplus b$.*

Preuve. Directement déduite du théorème 2.1.4. ■

Ainsi, les nimbers sont des infinitésimaux qui correspondant à des positions de jeu $\mathcal{N}$ et ne sont donc pas comparables avec 0. Il en existe bien d'autres qui eux sont comparables avec 0, comme par exemple $\{0|*\}$ et $\{*|0\}$, qu'on notera respectivement $\uparrow$, et $\downarrow$. D'autres peuvent être construits par la somme, comme le jeu $\uparrow + *$ ou $\Uparrow = \uparrow + \uparrow$. Si certains infinitésimaux sont incomparables entre eux ($*$ et $\uparrow$), d'autres peuvent être comparés ($*2 < \uparrow$), permettant les simplifications d'écriture via les options dominées ou réversibles. Citons par exemple les égalités

$$\{\uparrow | \downarrow\} = \{\uparrow |0\} = \{0| \downarrow\} = *,$$

ou encore, si x est un nombre dyadique,

$$\{x|x\} = x + *.$$

Enfin, notons que la somme d'un nombre et d'un infinitésimal est dite *nombresque*, autrement dit une valeur infinitésimalement proche d'un nombre. Les infinitésimaux sont eux-mêmes des nombresques.

Les switches

Enfin, il existe d'autres jeux qui ne sont ni des nombres, ni des nombresques. C'est le cas des *switches*.

Définition 2.2.18 (switch)**.** *Un* switch *est un jeu de la forme* $\{x|y\}$*, où* x *et* y *sont des nombre dyadiques tels que* $x > y$*. En particulier, le switch* $\{x|-x\}$ *sera noté* $\pm x$*.*

Les switches sont des jeux où chaque joueur a envie de jouer en premier car laisser l'autre commencer conduirait à une position de valeur moins favorable.

La somme d'un switch et d'un nombre se calcule bien. En l'occurrence, si G est un switch $\{x|y\}$ et z est un nombre dyadique, alors on a :

$$\{x|y\} + z = \{x + z|y + z\}.$$

De manière générale, les switches et autres valeurs (obtenues par récursion avec des switches — par exemple $\{\pm 2|\{0|-2\}\}$) ou les sommes de switches avec des infinitésimaux sont difficiles à simplifier ou à comparer. Nous ne rentrerons ici pas dans les détails de leur traitement.

2.2.5 Comment jouer sur une somme de jeux ?

C'est la question principale qui nous anime, dès lors qu'on connaît les valeurs des composantes d'une somme. Par exemple, imaginons le jeu $3 + * \pm 2$, peut-on savoir qui gagne et quelle stratégie appliquer ?

Si on a vu que les sommes de nombres et de nimbers se simplifient bien et permettent de répondre favorablement à ces questions, cela devient plus compliqué lorsque l'on somme des jeux ayant des types de valeurs différentes. Pour savoir quelle composante est prioritaire sur quelle autre, Conway a défini la notion de *température* d'un jeu. Il s'agit d'une valeur numérique qui quantifie le degré d'urgence pour jouer dans ce jeu. Sans rentrer dans les détails de la définition de cette valeur ([122], Def. 5.2), on a les propriétés suivantes :

Proposition 2.2.19 (Température)**.** *Soit* $G \in \mathbb{G}$*.*

- *Si* G *est un nombre* $\frac{m}{2^n}$ *avec* $m \neq 0$*, alors sa température est négative et vaut* $-\frac{1}{2^n}$ *;*
- *si* G *est nombresque, alors sa température est nulle ;*
- *si* G *n'est ni un nombre, ni nombresque, alors sa température est positive. En particulier si* G *est un switch* $\{x|y\}$ *(avec* $x > y$*), alors sa température vaut* $\frac{1}{2}(x - y)$*.*

Dans une somme de switches, de nombresques et de nombres, la stratégie est alors la suivante : jouer à chaque tour dans le jeu ayant la plus haute température. Les nombres sont donc considérés comme des jeux où il n'y a aucune urgence de jouer puisque d'une certaine manière, ce seront des coups "gâchés". On dit qu'il s'agit de jeux *froids*. Au contraire, les switches sont dits *chauds* et les nombresques sont dits *tièdes*. Par exemple, le jeu $\{2|-\frac{1}{2}\}+\{1|-1\}+\{0|-1\}+\{0|0\}+\frac{3}{4}$ est une somme de jeux de températures respectives $\frac{5}{4}, 1, \frac{1}{2}, 0$ et $-\frac{1}{4}$. Si Left commence, on arrivera après 4 coups à la position de valeur

$$2-1+0+0+\frac{3}{4}=\frac{7}{4}.$$

Si Right commence, on obtiendra le jeu de valeur

$$-\frac{1}{2}+1-1+0+\frac{3}{4}=\frac{1}{4}.$$

Dans tous les cas, la résultante sera strictement positive et ainsi le jeu global est $\mathcal{L}$.

Dans les cas plus complexes (avec des valeurs à température positive qui ne sont pas des switches), d'autres notions sont mises en place pour savoir comment jouer (cf. la notion de thermographe) mais nous ne les aborderons pas ici. De manière générale, décider si Left gagne sur une somme de jeux sous forme canonique reste un problème PSPACE-complet [137].

2.3 Résolution pratique de jeux

Dans la partie précédente, nous avons vu comment étudier de manière théorique un jeu combinatoire. Nous allons maintenant considérer la résolution pratique de jeux. Nous nous concentrons ici sur des jeux impartiaux (les deux joueurs ont les mêmes coups à jouer). Comme vu précédemment, les valeurs ne peuvent alors être que des nimbers et une fois ceux-ci connus, la résolution du jeu est assez aisée. En pratique, calculer ces nimbers ou simplement calculer l'issue du jeu se révèle la plupart du temps compliqué. Dans un premier temps, nous montrons en quoi la classe de complexité PSPACE est la classe pertinente pour l'étude de jeux et donnons un exemple de jeux PSPACE-complet qui devient polynomial en changeant légèrement les règles. Nous nous intéressons ensuite à une classe importante de jeux impartiaux, les jeux octaux, pour lesquels un comportement périodique est conjecturé. Il s'agit selon Guy [42] de l'un des problèmes non résolus les plus importants des jeux combinatoires.

2.3.1 Quelle classe de complexité pour les jeux ?

Vérifier qu'une stratégie est gagnante nécessite souvent un temps de calcul exponentiel car nous n'avons guère d'autre choix que de tester toutes les options de jeu de l'adversaire. Par contre, lorsqu'un jeu ne boucle pas, il est en général possible de tester une stratégie avec un espace de taille polynomial en la taille du jeu [1]. C'est principalement pour cette raison que la classe de complexité pertinente pour l'étude des jeux est la classe PSPACE, contenant tous les problèmes de décision pouvant se résoudre sur une machine de Turing avec un espace de taille polynomial. Un problème est PSPACE-complet s'il est dans PSPACE et que tout problème de PSPACE peut s'y réduire en temps polynomial.

Le problème canonique PSPACE-complet est le problème QBF — *Quantified Boolean Formula* [5]. Il s'agit de décider si une formule de la forme $Q_1 x_1 Q_2 x_2 \cdots Q_n x_n \varphi(x_1, \ldots, x_n)$ est vraie, où les Q_i sont des quantificateurs, x_i des variables booléennes et φ est une formule close, c'est-à-dire que les seules variables apparaissant dans φ sont $x_1, \ldots, x_n$.

La nature de ce problème est exactement la même que celle de chercher une stratégie gagnante pour un jeu combinatoire. En effet, un certificat pour le problème QBF ressemble fort à un arbre de jeu donnant la stratégie gagnante d'un joueur. De manière plus formelle, on peut considérer le jeu QBF-GAME. Le plateau de ce jeu est une formule booléenne φ contenant des variables booléennes $x_1, \ldots, x_n$. Le premier joueur choisit la valeur de la première variable x_1. Puis le deuxième joueur fait de même avec x_2 et ainsi de suite. Le premier joueur gagne si et seulement si, à la fin, φ est vraie. Le problème est de décider l'issue de ce jeu : le premier joueur a-t-il une stratégie gagnante ? Clairement, le premier joueur a une stratégie gagnante si et seulement si la formule

$$\exists x_1 \forall x_2 \exists x_3 \cdots \varphi(x_1, x_2, \ldots, x_n)$$

est vraie. On peut démontrer assez facilement que ce problème est équivalent à QBF et donc que QBF-GAME est PSPACE-complet. Ainsi tout problème PSPACE-complet est dans sa nature proche d'un jeu.

Un des premiers jeux à avoir été démontré comme PSPACE-complet est le jeu GEOGRAPHY. C'est d'ailleurs à partir de ce jeu que Lichtenstein et Sipser ont montré que le Go dans une version généralisée était PSPACE-difficile [79]. GEOGRAPHY se joue sur un graphe orienté. Un jeton est déposé sur un sommet s du graphe. Puis les joueurs déplacent l'un après l'autre le

1. On considère en général pour la taille du jeu un encodage de la position initiale.

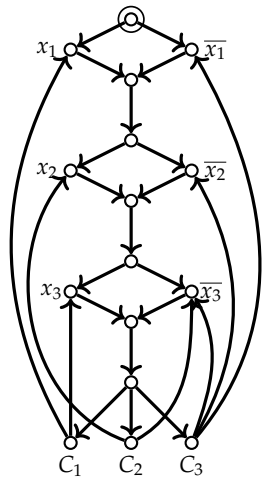


FIGURE 2.4 – *Réduction de* QBF-GAME *vers* GEOGRAPHY *pour la formule* $\exists x_1 \forall x_2 \exists x_3, (x_1 \vee x_3) \wedge (x_2 \vee \overline{x_3}) \wedge (\overline{x_1} \vee \overline{x_2} \vee \overline{x_3})$.

jeton en suivant les arcs et sans repasser par un arc déjà visité [2]. Le joueur ne pouvant plus jouer a perdu.

Théorème 2.3.1 (Schaeffer [120])**.** *Décider si le premier joueur a une stratégie gagnante dans* GEOGRAPHY *est* PSPACE*-complet.*

Preuve. La réduction se fait à partir de QBF-GAME joué sur des formules avec un nombre impair de variables et sous forme normale conjonctive (on se convaincra assez facilement que cela ne change pas la complexité du problème). À partir d'une telle formule, un graphe est construit de telle sorte que le premier joueur a une stratégie gagnante dans un des jeux si et seulement s'il a une stratégie gagnante dans l'autre jeu. Un exemple de construction est donné dans la figure 2.4 pour la formule $\exists x_1 \forall x_2 \exists x_3, (x_1 \vee x_3) \wedge (x_2 \vee \overline{x_3}) \wedge (\overline{x_1} \vee \overline{x_2} \vee \overline{x_3})$. Le premier joueur peut "décider" les valeurs des variables x_1 et x_3 en orientant le jeton vers le littéral correspondant tandis que le joueur 2 décide la valeur de x_2 et la clause finale. Le premier joueur gagne s'il peut revenir sur un littéral qui a été choisi, le deuxième joueur ne pouvant plus jouer. Cela signifie aussi que la clause est satisfaite. ■

2. L'origine de ce jeu vient du jeu populaire où l'on doit dire chacun son tour le nom d'un pays commençant par la dernière lettre du pays précédemment cité, d'où le nom GEOGRAPHY !

Cette preuve s'adapte facilement si l'on ne peut pas revenir sur un sommet déjà ou bien si l'on joue sur un graphe planaire [79]. Par contre, la variante suivante, jouée sur un graphe orienté, est polynomiale. Le jeu UV-GEOGRAPHY (Undirected-Vertex Geography) se joue sur un graphe non orienté et au lieu de ne pas réutiliser d'arcs, les joueurs ne peuvent pas réutiliser un sommet déjà visité. Rappelons qu'un *couplage* dans un graphe est une union d'arêtes deux à deux non adjacentes. Il *sature* un sommet si l'une des arêtes est adjacente à ce sommet.

Théorème 2.3.2 ([36])**.** *Le premier joueur gagne à* UV-GEOGRAPHY *sur* (G, s) *si et seulement si tout couplage de taille maximum de* G *sature* s*.*

Preuve. Supposons qu'il existe un couplage M de taille maximum qui ne sature pas s. Alors le second joueur joue à chaque fois sur une arête de M et gagne. Sinon, le premier joueur choisit un couplage M de taille maximum et joue sur ce couplage. Si le second joueur gagne, les arêtes formées par ses coups forment un couplage de taille maximum qui ne sature pas s. ■

Cela implique un algorithme polynomial pour décider si le premier joueur gagne à UV-GEOGRAPHY. En effet, on peut utiliser l'algorithme d'Edmonds [32] qui calcule en temps polynomial la taille maximum d'un couplage dans un graphe. Il suffit alors de comparer les tailles maximum des couplages dans G et $G \setminus \{s\}$.

Le cas de UV-GEOGRAPHY reste assez exceptionnel et la plupart des jeux combinatoires connus (au sens large) sont PSPACE-difficiles lorsque l'on généralise leurs règles sur des plateaux de taille quelconque [3]. C'est le cas de Hex [33], du Go [79] ou des échecs [35]. Ces deux derniers sont même EXPTIME-complet (du fait des boucles possibles dans les règles du jeu). Pour de nombreux jeux, la complexité reste un problème ouvert important. C'est par exemple le cas de ARC KAYLES et DOMINEERING. Pour en savoir plus, nous invitons le lecteur à consulter le tableau de complexité des jeux maintenu par Kyle Burke [4].

2.3.2 Conjecture de Guy sur les jeux octaux

Il existe tout de même une classe importante de jeux pour lesquels on conjecture une résolution polynomiale. Il s'agit des *jeux octaux* qui

3. On peut généraliser les règles du jeu d'échecs en prenant un plateau carré de taille $n \times n$ et les pièces habituelles en respectant les mêmes proportions que sur le 8×8. Le problème est alors de décider, pour une configuration quelconque, si l'un des joueurs a une stratégie gagnante.

4. `http://turing.plymouth.edu/~kgb1013/rulesetTable.php`

généralise à la fois le jeu de NIM et le jeu CRAM (DOMINEERING où les joueurs peuvent jouer les dominos dans n'importe quelle direction), joué sur une ligne.

Les jeux octaux se jouent avec plusieurs tas de jetons. Chacun son tour, les joueurs prennent des jetons dans un seul des tas et peuvent éventuellement séparer le tas en 2. Les règles de prises sont déterminées par le nom du jeu, qui est un code noté $\cdot\mathbf{d_1 d_2 d_3}\cdots$ où $\mathbf{d_i}$ est un entier compris entre 0 et 7. Chaque d_i étant compris entre 0 et 7, on peut écrire de manière unique $\mathbf{d_i} = x_0^i + 2x_1^i + 4x_2^i$ avec $x_j^i \in \{0,1\}$. La règle de prise est la suivante : on peut retirer i jetons dans l'un des tas et séparer le tas en j tas si et seulement si $x_j^i = 1$. Le joueur qui ne peut plus retirer de jetons a perdu.

Par exemple, $\mathbf{d_2} = 3$ indique qu'on peut enlever deux jetons d'un tas, sans le diviser en deux, tandis que $\mathbf{d_2} = 7$ indique que l'on peut enlever deux jetons en divisant éventuellement le tas en deux. Le jeu de NIM a pour code $\cdot\mathbf{3333}\cdots$ tandis que jouer à $\cdot\mathbf{07}$ revient à jouer à CRAM [5]. En effet, on peut considérer les jetons comme étant alignés. On ne prend alors que des jetons consécutifs et les trous délimitent les tas de jetons.

Pour résoudre complètement un jeu octal, il suffit de connaître sa valeur de Conway avec un seul tas de jetons. En effet, les jeux octaux sont des jeux impartiaux, leurs valeurs sont donc des nimbers. Ainsi, dès qu'il y a plusieurs tas, comme il s'agit en fait d'une somme de jeux d'un seul tas, on peut déduire la valeur du jeu en utilisant le théorème 2.2.17. On considère donc un jeu octal comme complètement résolu lorsque l'on connaît la séquence de ses nimbers $(u(n))_{n\in\mathbb{N}}$ pour un tas de taille n, appelée *séquence de Grundy*. Pour calculer $u(n)$ pour n fixé, il suffit de calculer les nimbers de toutes les options obtenues en jouant sur un tas de taille n et de prendre le plus petit nimber non présent (Théorème 2.2.16).

La séquence de Grundy de NIM est ainsi $*0, *1, *2, *3, *4, *5, \ldots$ que l'on notera $012345\cdots$ pour plus de lisibilité. D'autres jeux octaux ont des séquences particulièrement simples. C'est le cas du jeu $\cdot\mathbf{333}$ qui correspond au jeu populaire des allumettes : chacun son tour, les joueurs prennent 1,2 ou 3 allumettes et la personne qui prend la dernière allumette à gagner. Il est assez rapide de voir que la séquence de Grundy de $\cdot\mathbf{333}$ est $01230123\cdots$. Cette séquence est périodique, comme c'est le cas pour tous les jeux qui s'écrivent avec un nombre fini de **0** et de **3**, appelés *jeux de soustraction* (car on ne peut que soustraire des jetons d'un tas et jamais diviser les tas en deux). On peut démontrer la périodicité en remarquant qu'il y a un nombre borné d'options dans chaque position. D'autres jeux octaux ont des séquences d'apparence plus compliquée, comme celle de $\cdot\mathbf{07}$ qui commence

5. Lorsqu'il y a un nombre fini de $\mathbf{d_i}$ non nuls, l'écriture s'arrête au dernier $\mathbf{d_i}$ non nul.

par

$$0011203110332240522 33\cdots.$$

Malgré son apparence aléatoire, elle est en fait périodique, de période 34. Pour démontrer qu'un tel jeu est périodique de période p à partir de l'indice i, il suffit de montrer que la périodicité est vraie jusqu'aux indices $2i + 2p + t$ où t est le plus grand nombre de jetons que l'on peut prendre [9]. De nombreux jeux octaux finis (c'est-à-dire avec un nombre fini de $\mathbf{d_i}$ non nuls) ont été étudiés et, de fait, tous ceux dont les calculs ont abouti ont une séquence de nimbers ultimement périodique, ce qui a mené à la conjecture suivante :

Conjecture 2.3.3 (Conjecture de Guy [9])**.** *La séquence de Grundy d'un jeu octal fini est ultimement périodique.*

Si cette conjecture est vraie, il suffirait alors, pour résoudre un jeu octal, de calculer le reste de n par la période pour savoir si la situation est gagnante ou perdante avec n jetons et cela donnerait aussi le coup à jouer. Cependant la période peut-être grande (pour $\cdot\mathbf{16}$ la période est de taille 149159 et la prépériode de taille 105351) et la conjecture loin d'être prouvée (l'étude de $\cdot\mathbf{06}$ est encore ouverte !).

Mentionnons pour finir des extensions des jeux octaux dans les graphes qui apportent de nouvelles questions de recherche. En effet, un jeu octal peut être vu comme un jeu sur un chemin où l'on retire des sommets suivant certaines règles. Il est ainsi naturel d'étudier des jeux similaires sur les graphes. Le jeu NODE KAYLES se joue sur un graphe simple non orienté. Chacun son tour, les joueurs choisissent un sommet et retirent du graphe ce sommet et tous ses voisins. Sur le chemin, cela revient à jouer à $\cdot\mathbf{137}$: en effet, dans $\cdot\mathbf{137}$, on peut retirer un sommet s'il est tout seul, ou bien deux sommets sur un bord (sans séparer le chemin) ou bien trois sommets adjacents n'importe où. Sans surprise, ce jeu est PSPACE-complet [120]. La réduction est moins simple que pour GEOGRAPHY. En effet, dans GEOGRAPHY, le nombre de coups jouables est le nombre de voisins et peut donc être très petit alors que dans NODE KAYLES, un joueur peut jouer sur n'importe quel sommet du graphe. Pour contrer cela, la réduction crée des sommets sur lesquels un joueur n'a jamais intérêt à jouer car il perdrait immédiatement. Notons que nous ne connaissons pas la complexité de NODE KAYLES sur les arbres.

Le jeu ARC KAYLES se joue sur un graphe où chacun son tour un joueur enlève une arête et les arêtes adjacentes. Ce jeu correspond à $\cdot\mathbf{07}$ sur les chemins. Sur une grille, cela correspond à placer des dominos sans superposition (comme à DOMINEERING mais en version impartiale : les

joueurs n'ont pas d'orientation prédéfinie et peuvent jouer les dominos dans le sens qu'ils veulent). Ce jeu appelé CRAM n'est pas résolu sur les grilles dont les côtés sont tous les deux impairs (lorsqu'un côté est pair, un argument de symétrie donne une stratégie évidente pour l'un des deux joueurs). La complexité générale d'ARC KAYLES n'est pas non plus connue.

D'autres généralisations des jeux octaux et en particulier des jeux de soustraction sur les graphes commencent aussi à être étudiées.

Chapitre 3

Jeux, topologie et automates

Jacques Duparc

Les jeux à deux joueurs et information parfaite sont un outil pour l'analyse des processus d'interaction. Lorsque les parties sont de longueur finie, l'un des joueurs possède toujours une stratégie pour l'emporter quoi que fasse son adversaire. Bien qu'en général cette propriété disparaisse lorsqu'on considère des parties infinies, dans certains cas néanmoins, de tels jeux infinis fournissent un puissant instrument permettant d'établir la complexité de nature topologique des langages de mots infinis reconnus par automates.

3.1 Introduction

Les jeux fournissent un cadre mathématique pour l'interprétation des phénomènes d'interaction. Des machines alternantes aux systèmes réactifs, en passant par la sémantique des jeux, de nombreux aspects de l'informatique s'expriment sous cette forme [4]. Ce chapitre s'intéresse à un usage très particulier des jeux : celui d'élucider la complexité topologique des langages de mots infinis reconnus par automates.

Dans la suite, les termes « mot » et « suite » seront utilisés de manière interchangeable. On notera A^* et A^ω respectivement pour l'ensemble des mots finis et des mots infinis sur un alphabet A. La concaténation des mots u et v sera notée uv. Les lettres a, b seront réservées pour les lettres de l'alphabet, u, v pour les mots finis et $\vec{a}, \vec{b}$ pour les mots infinis.

Un automate de Büchi [106] est de la forme $\mathscr{A} = (A, Q, q_i, \Delta, F)$ où

1. A est un alphabet fini ;
2. Q est un ensemble fini d'états ;

3. q_i est l'état initial ;
4. $\Delta \subseteq Q \times A \times Q$;
5. $F \subseteq Q$ désigne l'ensemble des états acceptants.

Il est déterministe lorsque Δ est une fonction $Q \times A \to Q$; ce qui signifie que pour tout couple $(q, a) \in Q \times A$ il existe un unique $q' \in Q$ tel que $(q, a, q') \in \Delta$. L'automate $\mathscr{A}$ accepte un mot infini $\vec{a} = a_0 a_1 a_2 \cdots$ s'il existe une exécution [1] $\rho_{\vec{a}} \in Q^\omega$ visitant infiniment souvent un état acceptant [2]. Les automates de parité [3] sont définis de manière identique, à l'exception de la condition d'acceptation qui substitue à l'ensemble F une fonction $c : Q \to \mathbb{N}$ et stipule qu'un mot infini $\vec{a}$ est accepté s'il existe une exécution $\rho_{\vec{a}}$ telle que $\limsup_{n\to\infty} c\left(\rho_{\vec{a}}(n)\right)$ est paire [106, 38]. Autrement dit $\vec{a}$ est accepté s'il existe une exécution telle que l'ensemble S des états visités infiniment souvent vérifie $\max\{c(q) \mid q \in S\}$ est un entier pair.

Le langage reconnu par un automate désigne l'ensemble des mots qu'il accepte. Automates de parité, automates de Büchi et automates de parité déterministes reconnaissent tous trois les mêmes langages dénommés *ω-réguliers*. Il est à noter que si $\mathscr{A} = (A, Q, q_i, \delta, c)$ est un automate de parité déterministe, alors le complémentaire du langage reconnu par $\mathscr{A}$ est reconnu par l'automate $\mathscr{A}^{\complement} = (A, Q, q_i, \delta, c')$ où c' est définie par $c'(n) = c(n) + 1$.

Enfin, on utilisera souvent la définition ensembliste de la notion d'arbre : un arbre T sur un alphabet A est un ensemble de mots finis $T \subseteq A^*$ clos par préfixes.

3.2 Jeux finis à deux joueurs et information parfaite

Nous allons considérer des jeux bien particuliers : il y a *deux joueurs* ; les parties sont *séquentielles* [4] et s'achèvent après un nombre de coups fini ; l'information est *parfaite* : à tout moment la configuration entière de la partie est accessible aux deux joueurs (rien n'est caché) [5] ; il y a un *gagnant*

1. Une exécution $\rho_{\vec{a}} = \rho_{\vec{a}}(0)\rho_{\vec{a}}(1)\rho_{\vec{a}}(2)\cdots$ doit vérifier $\rho_{\vec{a}}(0) = q_i$ et pour tout entier n, $(\rho_{\vec{a}}(n), a_n, \rho_{\vec{a}}(n+1)) \in \Delta$.

2. L'Exemple 3.3.21 page 75 offre un automate de Büchi reconnaissant le langage $(\{0,1\}^*1)^\omega$.

3. L'Exemple 3.3.19 page 74 et ceux qui suivent proposent des automates de parités reconnaissant divers langages sur l'alphabet $\{0,1\}$.

4. Il n'y a pas de coups simultanés, les joueurs jouent à tour de rôle, un joueur pouvant jouer plusieurs tours à la suite, mais les joueurs ne jouent jamais en même temps.

5. En particulier il n'y a pas d'intervention du hasard : les joueurs ne lancent pas de dé, ne tirent pas de carte, etc.

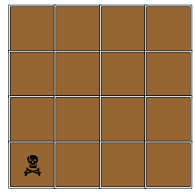
et un *perdant* [6]. Nous réservons d'ailleurs le mot « jeu » pour désigner de tels jeux.

Les jeux de hasard tel que le poker ou le jeu de l'oie ne sont pas de ce type. Pas plus que ne l'est le jeu de la bataille navale (il n'y a pas d'intervention du hasard, mais la configuration de la partie n'est pas pleinement accessible aux joueurs [7]).

Parmi les jeux de ce type on trouverait les versions du jeu d'échec et du jeu de dames s'il n'y avait pas de match nul [8].

Exemple 3.2.1. *Le jeu de la tablette de chocolat est un jeu à deux joueurs (***0** *et* **1***) qui jouent à tour de rôle. Le joueur* **0** *débute et les joueurs doivent à chaque fois manger un carreau de chocolat de la tablette. Mais lorsqu'il mange le carreau* (i, j)*, le joueur doit également croquer tous les carreaux* (i', j') *tels que* $i' \geqslant i$ *et* $j' \geqslant j$*. Malheureusement, le carreau* $(0, 0)$ *est mortel. Par conséquent celui qui mange le dernier carreau perd la partie.*

Ainsi, si la tablette initiale est carrée avec 16 carreaux de chocolat :



au premier coup, le joueur **0** *a exactement 16 coups possibles, puis le joueur* **1** *en a autant que de carrés de chocolat que lui a laissés son adversaire, etc. Ce jeu est séquentiel (puisqu'il est alterné), à deux joueurs et information parfaite. Tout comme la plaque de chocolat, il est fini et l'un des joueurs gagne (il survit !) tandis que l'autre perd (il meurt !). Il tombe dans le cadre des jeux que nous étudions.*

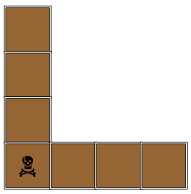
Si le joueur **0** *procède de la manière suivante :*

1. *au premier coup il choisit le carreau* $(1, 1)$*, en sorte qu'il laisse son adversaire dans la configuration suivante :*

6. Il n'y a pas de match nul et il n'y a rien d'autre à gagner que le gain de la partie elle-même.

7. C'est précisément ce qui est en jeu ici : deviner quelles sont les positions des navires de son adversaire.

8. Par exemple, si l'on change la définition du jeu d'échec pour convenir maintenant que les *Blancs* l'emportent en cas de match nul dans la version officielle, alors ce jeu est bien un jeu séquentiel — il est même alterné — à deux joueurs et information complète. De plus il est fini puisque toute partie se déroule en un nombre de coups fini. En effet, il existe au moins deux conditions qui assurent la finitude d'une partie, puisque celle-ci est déclarée nulle : lorsque 50 coups sont joués sans prise ni mouvement de pion ; ou lorsqu'une même configuration se répète par trois fois.



2. *ensuite, à chaque fois qu'il doit jouer, lorsque son adversaire choisit le carreau* $(0, i)$*, resp.* $(i, 0)$*, le joueur* **0** *choisit le carreau symétrique* $(i, 0)$*, resp.* $(0, i)$*.*

On voit alors aisément que rétablissant ainsi toujours la symétrie de la forme « en L » de (l'état actuel de) la plaque de chocolat, le joueur **0** *assure que son adversaire mange le carreau* $(0, 0)$ *et meurt. Nous venons de montrer que le joueur* **0** *— celui qui débute la partie — possède une manière de jouer — dépendant des coups de son adversaire — qui lui assure à tous les coups la victoire*[9]*. On appelle cela une* stratégie gagnante*.*

L'ensemble de tous les coups possibles d'un tel jeu fini est représenté par un arbre étiqueté démuni de branche infinie. Chaque nœud correspond à une configuration du jeu — la racine étant étiquetée par la configuration initiale : celle dans laquelle le jeu débute. Chaque branche — de la racine vers une feuille — représente une partie possible.

Ainsi, si l'on prend comme exemple le jeu d'échec, la configuration initiale représentera la disposition initiale des pièces sur l'échiquier avec la mention que c'est au tour des *Blancs* de jouer. Les fils de cette configuration initiale seront toutes les configurations que les *Blancs* peuvent atteindre en un coup (il y en a vingt)[10]. Ensuite chacun des fils de la configuration initiale possède également vingt fils[11]. Cela fait donc quatre cents petits fils pour la racine ; l'arbre continuant de croître de manière exponentielle.

Définition 3.2.2. *Un* arbre de jeu fini (T, e) *est un arbre non vide sans branche infinie* T*, étiqueté par la fonction* $e : T \to \{\mathbf{0}, \mathbf{1}\}$*. Le jeu à deux joueurs et information parfaite associé à* (T, e) *consiste alors à*

— *placer un jeton sur la racine de l'arbre, puis*

9. Notons toutefois que notre démonstration ne vaut que si la tablette comporte au moins quatre carreaux de chocolat. Dans le cas où elle n'en comporte qu'un, celui-ci est nécessairement empoisonné et le joueur **0** perd immédiatement la partie au premier coup.

10. Chaque pièce (il y en a huit) peut être déplacée vers l'avant soit d'une case, soit de deux cases ; de plus, chacun des *Cavaliers* peut être sorti soit vers la droite, soit vers la gauche.

11. Les possibilités de coups pour les *Noirs* sont les mêmes que les *Blancs* avaient à l'étape précédente.

— *pour chaque nœud n sur lequel le jeton se trouve, le joueur $e(n)$ pousse le jeton sur l'un des fils du nœud n de son choix, mais perd la partie si n est une feuille.*

En théorie des Jeux, on parle de jeu représenté sous *forme extensive* plutôt que d'arbre de jeu.

Intuitivement, le déroulement d'une partie correspond à la production d'une branche de l'arbre et le joueur qui ne peut avancer plus avant dans l'arbre perd la partie. Le fait que l'arbre de jeu n'ait pas de branche infinie garantit que toute partie est finie. L'étiquetage de l'arbre a deux fonctions : pour les nœuds intérieurs il indique à quel joueur c'est le tour de jouer, et pour les feuilles celui qui perd la partie — Pour le jeu de la tablette de chocolat (Exemple 3.2.1) avec une tablette carrée comportant 16 carreaux de chocolat, l'arbre de jeu associé aura alors une racine dotée de 16 fils (un pour chaque configuration de la tablette après que le premier joueur ait joué), puis chacun de ces fils aura lui-même exactement le même nombre de fils qu'il reste de carreaux de chocolat dans la configuration qu'il représente.

Exemple 3.2.3. *La partie la plus longue représentée par cette arbre comporte quatre coups :*

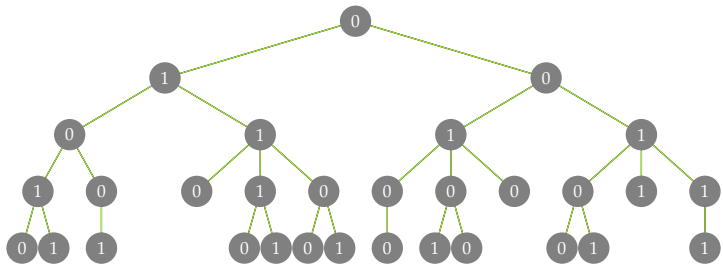


FIGURE *3.1 – Un arbre de jeu.*

Définition 3.2.4. *Une* stratégie *pour le joueur* **0** *dans le jeu associé à un arbre de jeu fini (T, e) est un arbre non vide étiqueté de la forme (σ, e) vérifiant $\sigma \subseteq T$, toute feuille de σ est également une feuille de T et pour tout nœud $n \in \sigma$ qui n'est pas une feuille :*

— *si $e(n) = 0$, alors un unique fils de n appartient à σ ;*

— *si $e(n) = 1$, alors tous les fils de n appartiennent à σ.*

On définit mutatis mutandis *une stratégie pour le joueur* **1** *en échangeant* **0** *et* **1**.

Une stratégie pour un joueur **J** induit en quelque sorte un sous-jeu du jeu initial qui fait qu'à chaque fois que c'est au tour du joueur **J** d'avancer,

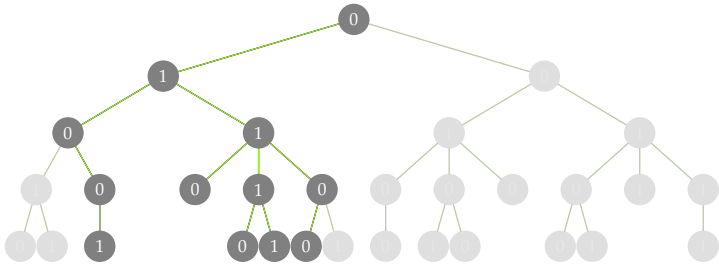


FIGURE 3.2 – *Une stratégie pour le joueur* **0** *dans le jeu de la Figure 3.1.*

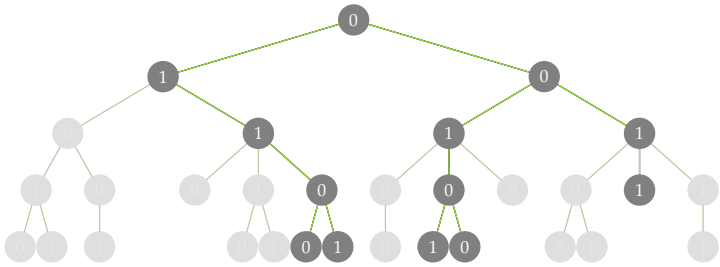


FIGURE 3.3 – *Une stratégie pour le joueur* **1** *dans le jeu de la Figure 3.1.*

un seul [12] des choix qui lui étaient auparavant possibles lui est maintenant autorisé ; mais par contre, à chaque fois que c'est au tour de son adversaire d'avancer, celui-ci conserve toutes les possibilités qu'il avait dans le jeu initial.

On dit qu'un joueur applique une stratégie si le jeu est restreint à celle-ci. Dès lors, si le joueur **0** applique une stratégie σ et le joueur **1** applique une stratégie τ, le jeu se restreint à une unique partie qui est l'arbre doté de la seule branche : $\sigma \cap \tau$.

Exemple 3.2.5. *Les arbres ci-dessous sont des exemples de stratégies respectivement pour le joueur* **0** *et pour le joueur* **1** *dans le jeu de la Figure 3.1.*

Définition 3.2.6. *Dans le jeu associé à un arbre de jeu fini* (T, e)*,*

— *une* stratégie σ *pour le joueur* **0** *est dite* gagnante *si pour toute feuille* $f \in \sigma$, $e(f) = 1$;

— *une* stratégie τ *pour le joueur* **1** *est dite* gagnante *si pour toute feuille* $f \in \tau$, $e(f) = 0$.

Ainsi, une stratégie pour un joueur **J** est gagnante si en l'appliquant scrupuleusement, ce joueur est certain de l'emporter quoi que fasse son adversaire. Une conséquence immédiate des jeux finis à deux joueurs et

12. Ou aucun dans le cas où l'on se trouve dans une feuille.

information parfaite est qu'au plus l'un des joueurs peut disposer d'une stratégie gagnante. Sinon, les deux joueurs appliquant leurs prétendues stratégies gagnantes respectives emporteraient tous les deux la partie. Or il n'y a qu'un gagnant.

On remarquera qu'aucune des stratégies proposées dans l'Exemple 3.2.5 n'est gagnante. Nous nous intéressons maintenant aux conditions qui garantissent l'existence d'une stratégie gagnante.

Définition 3.2.7. *Un jeu est* déterminé *si l'un des deux joueurs possède une stratégie gagnante.*

Pour une certaine classe de jeux, le fait que ceux-ci soient déterminés est une assertion très forte. Elle transforme un jugement négatif (« le joueur **J** n'a pas de stratégie gagnante ») en un jugement positif (« le joueur $\mathbf{1} - \mathbf{J}$ possède une stratégie gagnante »). Autrement dit, l'absence d'un sous-arbre [13] d'un certain type entraîne l'existence d'un arbre d'un autre type. On voit dès lors que le principe de détermination n'est pas constructif : il est affirmé qu'une stratégie gagnante existe pour un joueur, sans qu'il soit donné de moyen de la construire comme nous allons le voir dans l'exemple qui suit.

Exemple 3.2.8. *Reprenons le jeu de la tablette de chocolat de l'Exemple 3.2.1 mais cette fois-ci au lieu de considérer une tablette carrée, nous allons nous intéresser au cas plus général d'une tablette rectangulaire* (n, m)*. Le carreau de chocolat empoisonné se trouve toujours en position* $(0, 0)$ *comme indiqué dans la figure ci-dessous.*

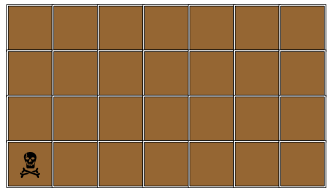


FIGURE *3.4 – Une tablette de chocolat.*

Nous savons que si la tablette est carrée, alors le joueur qui débute la partie — le joueur **0** *— possède une stratégie gagnante excepté dans le seul cas où la tablette ne contient qu'un unique carreau de chocolat (nécessairement empoisonné). En supposant que ce jeu est* déterminé *— ce que confirmera le Théorème 3.2.9 — nous allons montrer que le joueur* **0** *possède une stratégie gagnante quelle que soit la taille* $(n, m) \neq (0, 0)$ *de la tablette de chocolat.*

13. Au sens de l'inclusion entre arbre et sous-arbre.

Puisque ce jeu est déterminé, pour montrer que le joueur **0** *possède une stratégie gagnante, il suffit de montrer que son adversaire n'en possède pas. Procédant par l'absurde, nous supposons que le joueur* **1** *possède une stratégie gagnante* τ *et construisons une stratégie gagnante* σ *pour le joueur* **0**.

Comme indiqué dans la figure en page 63, nous considérons deux parties jouées : la partie G (pour gauche) et D (pour droite) ;

- *dans la partie G, le joueur* **1** *applique sa stratégie gagnante* τ *et va donc tout naturellement gagner cette partie ;*
- *dans la partie D, le joueur* **1** *joue de manière totalement libre. Le joueur* **0** *applique la stratégie* σ *que nous décrivons maintenant.*

Nous notons G_i, D_i *les* i^{es} *coups dans les parties respectives G et D, et définissons* σ *comme suit :*

- G_0 *: le joueur* **0** *mange le carré* (n, m) *;*
- G_{2i+1} *est la réponse par* τ *au coup* G_{2i} *du joueur* **0** *;*
- D_{2i} *est la copie par le joueur* **0** *du coup de* **1** *en* G_{2i+1} *(ainsi, le premier coup du joueur* **0** *dans la partie D consiste à manger tous les chocolats qui ont été mangés dans la partie G au premier coup par son adversaire et par lui-même) ;*
- D_{2i+1} *est le libre choix du joueur* **1**.

Puisque τ *est une stratégie gagnante pour le joueur* **1**, *celui-ci gagne la partie G. C'est donc son adversaire qui mange le carré de chocolat empoisonné. Mais dans la partie G, les carrés de chocolat mangés par le joueur* **1** *sont — à l'exception du carré* $(0, 0)$ *— exactement les mêmes que ceux mangés par le joueur* **0** *dans la partie D. C'est donc le joueur* **1** *qui perd la partie D, montrant ainsi que* σ *est une stratégie gagnante pour* **0**, *ce qui contredit que* τ *le soit le pour* **1**.

Le résultat qui suit nous permet de conclure que dans l'exemple précédent, l'un des joueurs possède une stratégie gagnante. Par conséquent, nous avons démontré que c'est le premier joueur qui possède une stratégie gagnante sans pourtant en proposer aucune.

Théorème 3.2.9. *Pour tout* arbre de jeu fini (T, e), *le jeu associé est déterminé.*

Preuve du Théorème 3.2.9. L'idée de la preuve de ce théorème est très simple. Elle repose sur ce que les théoriciens des jeux appellent « *backward induction* » et que l'on peut présenter sous la forme d'un algorithme de coloriage.

- On colorie en vert chacune des feuilles de l'arbre qui sont étiquetées par 1 et en rouge celles qui sont étiquetées par 0. Ainsi, une feuille est verte si et seulement si, pour le jeu commençant en cette feuille, le joueur **0** possède une stratégie gagnante ; elle est rouge si et seulement si, pour le jeu commençant en cette feuille, le joueur **1** possède une stratégie gagnante.

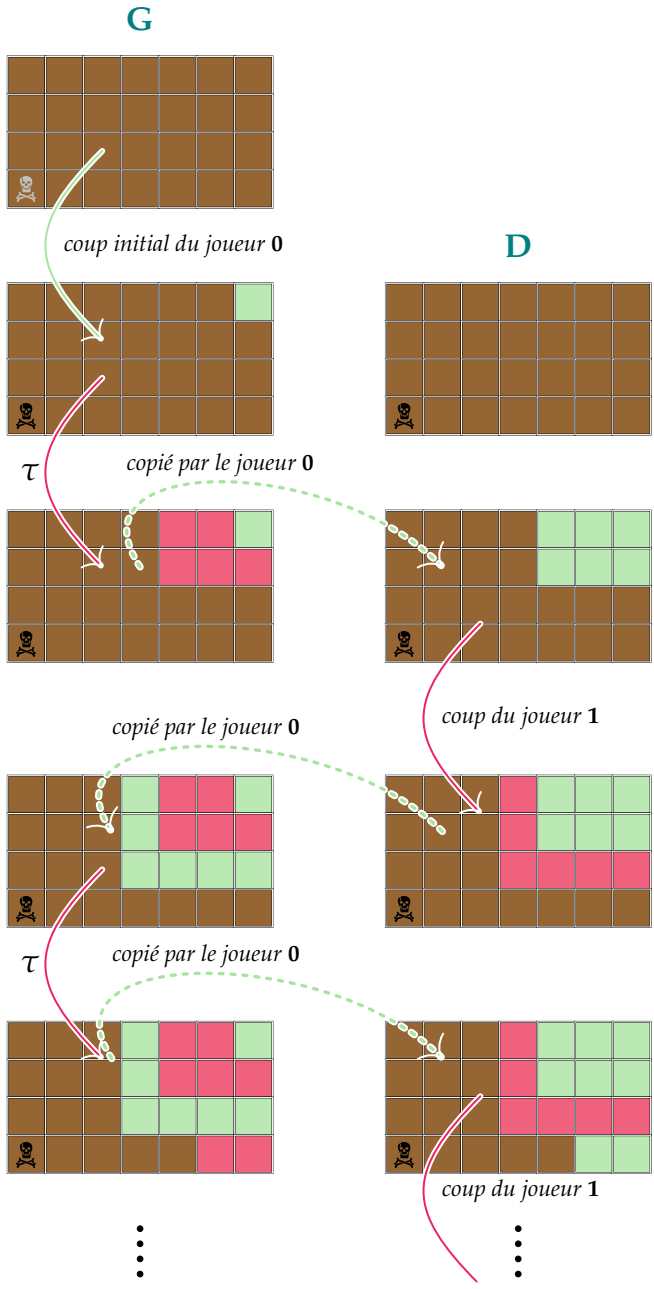


FIGURE 3.5 – *Stratégie gagnante pour* **0** *à partir d'une stratégie gagnante pour* **1**.

— Pour chaque nœud intérieur n, on colorie ce nœud en **vert** si

— soit n est étiqueté 0 et l'un de ses fils est colorié en vert ;
— soit n est étiqueté 1 et tous ses fils sont coloriés en vert.

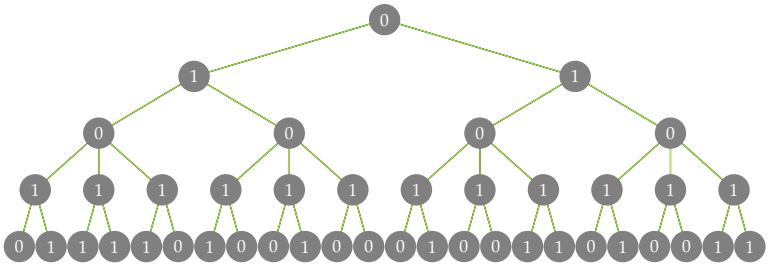
rouge si
— soit n est étiqueté 1 et l'un de ses fils est colorié en rouge ;
— soit n est étiqueté 0 et tous ses fils sont coloriés en rouge.

La raison de ce coloriage est la suivante : en un nœud n où le joueur **J** doit jouer, **J** possède une stratégie gagnante s'il existe au moins un des fils du nœud n où il possède une stratégie gagnante, puisqu'il peut alors choisir ce nœud et appliquer ensuite sa stratégie gagnante. Si pour chacun des fils du nœud n le joueur $\mathbf{1} - \mathbf{J}$ possède une stratégie gagnante, alors bien qu'au nœud n ce soit au tour de **J** de jouer, c'est néanmoins son adversaire qui possède une stratégie gagnante.

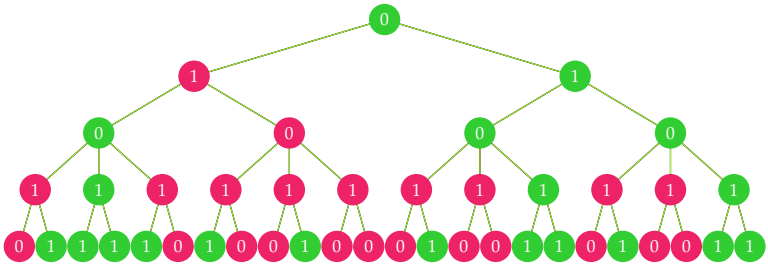
Finalement, la couleur de la racine — verte pour **0** et rouge pour **1** — indique lequel des deux joueurs possède une stratégie gagnante.

□ *Thm. 3.2.9*

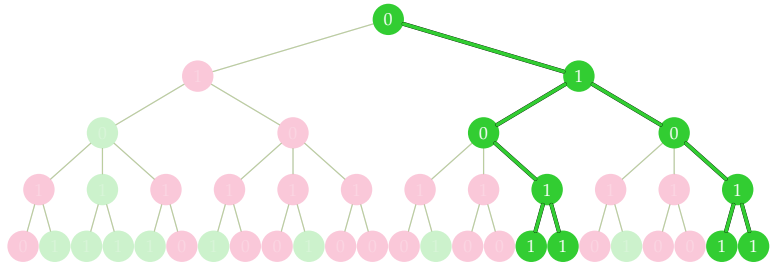
Exemple 3.2.10. *On considère le jeu ci-dessous :*



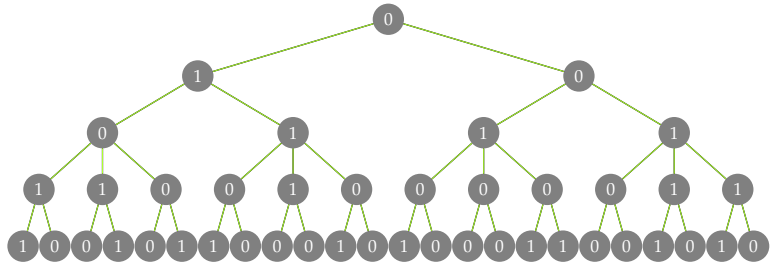
Auquel nous appliquons l'algorithme de coloriage :



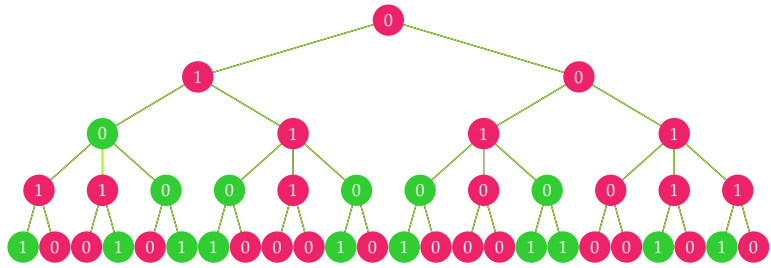
La racine étant verte, nous en déduisons une stratégie gagnante pour le joueur **0** *:*



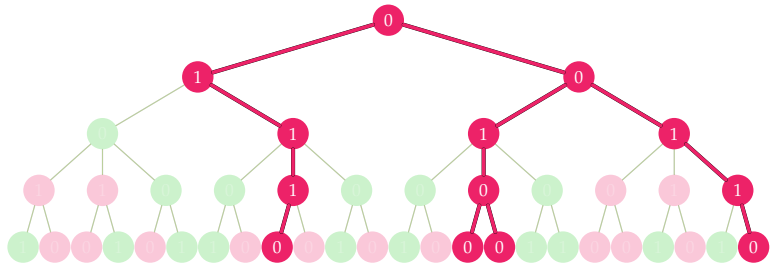
Exemple 3.2.11. *On considère le jeu suivant :*



Auquel nous appliquons l'algorithme de coloriage



La racine étant rouge, nous en déduisons une stratégie gagnante pour le joueur **1** *:*



Évaluation d'une formule

S'ils semblent abstraits, ces jeux sont en fait au cœur de la notion d'évaluation d'une formule : savoir si une formule est vraie ou fausse dans un modèle donné, revient à résoudre un jeu. C'est en particulier le cas pour la logique du premier ordre [22].

Définition 3.2.12. *Soient $\mathscr{L}$ un langage de la logique du premier ordre, $\mathscr{M}$ une $\mathscr{L}$-structure et ϕ une formule close de $\mathscr{L}$ dont les connecteurs sont parmi $\{\neg, \vee, \wedge\}$. On définit le* jeu d'évaluation *de la formule ϕ dans $\mathscr{M}$, noté $\mathbb{EV}(\phi, \mathscr{M})$ comme un jeu fini à deux joueurs — dénommés Vérificateur et Falsificateur — et information parfaite. Les coups sont définis par :*

si ϕ est	*c'est au tour de*	*on continue avec*
atomique	*personne*	*le jeu s'arrête*
$\exists x\, \psi$	*V choisit a dans le domaine de $\mathscr{M}$*	$\psi_{[a/x]}$
$\forall x\, \psi$	*F choisit a dans le domaine de $\mathscr{M}$*	$\psi_{[a/x]}$
$(\phi_0 \vee \phi_1)$	*V choisit $i \in \{0,1\}$*	ϕ_i
$(\phi_0 \wedge \phi_1)$	*F choisit $i \in \{0,1\}$*	ϕ_i
$\neg\psi$	*V et F échangent leurs rôles*	ψ

Par construction, le jeu s'arrête sur une formule atomique de la forme

$$R(t_1, \ldots, t_n)_{[a_1/x_1, \ldots, a_k/x_k]}$$

où $x_1, \ldots, x_k$ sont toutes les variables présentes dans $R(t_1, \ldots, t_n)$ et $a_1, \ldots, a_k$ sont des éléments du domaine de $\mathscr{M}$. Le Vérificateur gagne si et seulement si

$$\left(t_1{}^{\mathscr{M}}_{[a_1/x_1, \ldots, a_k/x_k]}, \ldots, t_n{}^{\mathscr{M}}_{[a_1/x_1, \ldots, a_k/x_k]}\right) \in R^{\mathscr{M}}.$$

Les règles de ce jeu sont définies pour obtenir le résultat suivant :

Théorème 3.2.13. *Si $\mathscr{L}$ est un langage de la logique du premier ordre, $\mathscr{M}$ une $\mathscr{L}$-structure et ϕ une formule close de $\mathscr{L}$ dont les connecteurs sont parmi $\{\neg, \vee, \wedge\}$, alors le Vérificateur possède une stratégie gagnante dans $\mathbb{EV}(\phi, \mathscr{M})$ si et seulement si la formule ϕ est vraie dans le modèle $\mathscr{M}$.*

3.3 Jeux infinis à deux joueurs et information parfaite

Le passage des jeux finis aux jeux infinis marque un saut. Tout devient moins à la fois moins simple et beaucoup plus technique puisque des notions topologiques sont requises.

Dans cette section nous considérons des jeux avec parties infinies. Parmi ceux-ci, le jeu de Gale-Stewart occupe une place centrale.

Définition 3.3.1. *Soit $L \subseteq A^\omega$, le jeu de Gale-Stewart $\mathcal{G}(L)$ est un jeu infini dans lequel les joueurs (I et II) choisissent à tour de rôle $a \in A$. Le joueur I débute la partie. Il emporte celle-ci si la suite $\vec{a}$ ainsi formée vérifie $\vec{a} \in L$. Dans le cas contraire, le joueur II gagne.*

FIGURE 3.6 – *Jeu de Gale-Stewart.*

Considérons tout d'abord, pour chaque entier non nul n, une version du jeu de Gale-Stewart $\mathcal{G}_n(M)$ pour $M \subseteq A^{2n}$ définie de manière analogue. Il est clair que tous ces jeux sont déterminés. Non seulement car ce sont des jeux finis à deux joueurs et information parfaite ; mais aussi, pour la raison que la formule qui exprime que le joueur I n'a pas de stratégie gagnante :

$$\neg\exists a_0 \forall a_1 \exists a_2 \forall a_3 \ldots \forall a_{2n-1}\, \vec{a} \in M$$

est *logiquement* équivalente à la formule

$$\forall a_0 \exists a_1 \forall a_2 \exists a_3 \ldots \exists a_{2n-1}\, \vec{a} \notin M$$

qui dit précisément que le joueur II possède une stratégie gagnante.

La détermination du jeu de Gale-Stewart peut alors être regardée comme la généralisation de cet argument à la pseudo « *formule infinie* » décrivant l'existence d'une stratégie gagnante pour le joueur I. En effet, la détermination stipule que si I n'a pas de stratégie gagnante, *i.e.*

$$\neg\exists a_0 \forall a_1 \exists a_2 \forall a_3 \ldots\ \vec{a} \in L,$$

alors le joueur II en possède une :

$$\forall a_0 \exists a_1 \forall a_2 \exists a_3 \ldots\ \vec{a} \notin L.$$

Toutefois, si la détermination des jeux à deux joueurs et information parfaite constitue un résultat élémentaire dans le cas des jeux finis, il en va tout autrement dans le cas des jeux infinis. D'une part on peut montrer qu'il existe des jeux non déterminés [14]. D'autre part, ces jeux sont effectivement déterminés pour une large classe d'ensembles (les *boréliens*).

14. Cela nécessite toutefois de faire appel à des hypothèses fortes comme l'*Axiome du Choix*.

3.3.1 Un jeu non déterminé

Il existe différentes manières de produire un jeu non déterminé, mais, peu ou prou, toutes nécessitent l'utilisation de l'*Axiome du Choix* [15]. Cet axiome de la théorie des ensembles affirme qu'étant donnée une famille d'ensembles non vides $\{E_i \mid i \in I\}$, il existe une fonction [16] $f : I \to \bigcup_{i \in I} E_i$ vérifiant $f(i) \in E_i$ pour chaque $i \in I$.

Avant d'utiliser cet Axiome du Choix pour obtenir un ensemble de gain qui serve notre but, nous définissons un jeu légèrement différent du jeu de Gale-Stewart : le jeu de Banach-Mazur.

Définition 3.3.2. *Soit $L \subseteq A^\omega$, le jeu de Banach-Mazur $\mathscr{B}(L)$ est identique dans les règles au jeu $\mathscr{G}(L)$ excepté que les joueurs jouent cette fois des suites finies* non vides *d'éléments de A au lieu d'éléments de A. Le joueur I emporte la partie si la concaténation $\vec{a}$ des suites ainsi jouées vérifie $\vec{a} \in L$. Dans le cas contraire, le joueur II gagne.*

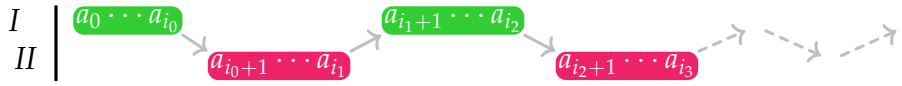


FIGURE 3.7 – *Jeu de Banach-Mazur.*

Il est très facile à partir d'ensembles quelconques A et $L \subseteq A^\omega$ de définir des ensembles A' et $L' \subseteq A'^\omega$ tels que le jeu $\mathscr{G}(L')$ simule le jeu $\mathscr{B}(L)$ en sorte que l'existence d'une stratégie gagnante pour l'un des joueurs dans le premier jeu induise l'existence d'une stratégie gagnante pour le même joueur dans le second. Il nous suffit donc pour montrer qu'il existe L' tel que $\mathscr{G}(L')$ ne soit pas déterminé, de produire un ensemble L tel que $\mathscr{B}(L)$ ne soit pas déterminé. L'Axiome du Choix va nous le procurer.

Définition 3.3.3. *$F \subseteq \{0,1\}^\omega$ est un* « flip set » *si pour tous $\vec{x}, \vec{y} \in \{0,1\}^\omega$, si $\exists k\ (x_k \neq y_k \wedge \forall n \neq k\ (x_n = y_n))$, i.e. $\vec{x}$ et $\vec{y}$ ne diffèrent que d'une valeur , alors $x \in F \iff y \notin F$.*

Proposition 3.3.4 (AC). *Si $F \subseteq \{0,1\}^\omega$ est un* « flip set », *alors le jeu $\mathscr{B}(F)$ n'est pas déterminé.*

15. L'Axiome du Choix en tant que tel n'est ni vrai ni faux. Il y a simplement des mathématiques qui le requièrent et d'autres non. Exactement comme il y a une géométrie euclidienne et d'autres qui ne le sont pas.

16. Cette fonction est appelée *fonction de choix* [46].

Preuve de la Proposition 3.3.4. Semblable à celle de l'Exemple 3.2.8, la preuve est par l'absurde; l'Axiome du Choix (AC) étant requis pour prouver l'existence d'un « *flip set* ». On suppose tout d'abord que le joueur *II* possède une stratégie gagnante τ et l'on montre que le joueur *I* en possède une également. La figure 3.8 représente deux parties de ce même jeu. Dans la partie inférieure le joueur *II* applique sa stratégie τ, lui garantissant la victoire. Dans la partie supérieure le joueur *I* applique une stratégie que l'on décrit maintenant contre le joueur *II* qui joue librement. On note $s_0 s_1 s_2 \cdots$ la suite des coups joués dans la partie supérieure et $i_0 i_1 i_2 \cdots$ celle des coups joués dans la partie inférieure.

Dans le partie inférieure, *I* débute avec la suite $i_0 = 0$ et *II* répond avec $i_1 = \tau(1)$ (dans la figure $i_1 = 10010$). Le joueur *I* joue alors la suite $s_0 = 1\tau(1)$ comme premier coup dans la partie supérieure, le joueur *II* y répondant par une suite s_1 (dans la figure $s_1 = 010001110$). A partir de là, chaque coup s_{n+1} jouée par *II* dans la partie supérieure est reporté par *I* dans la partie inférieure comme coup i_{n+2} et la réponse $i_{n+3} = \tau(\langle i_0 \cdots i_{n+2} \rangle)$ de *II* est copiée comme coup s_{n+2} de *I* dans la partie supérieure.

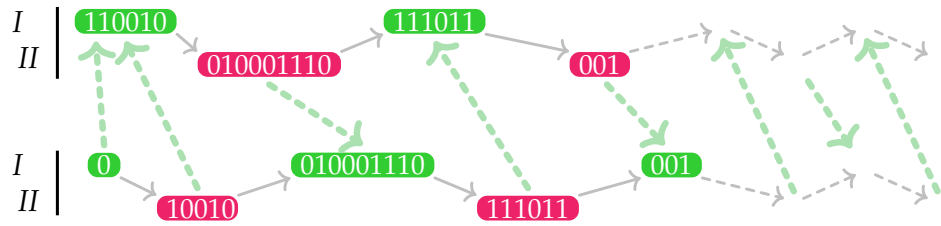


FIGURE 3.8 – *Stratégie d'imitation dans les jeux de Banach-Mazur.*

On remarque alors que la concaténation des coups joués dans les parties inférieure et supérieure engendrent deux suites infinies $\vec{i}, \vec{s} \in \{0,1\}^\omega$ qui ne diffèrent que par leurs premières valeurs. Puisque dans la partie inférieure *II* appliquait τ, on obtient $\vec{i} \notin F$. D'où $\vec{s} \in F$ puisque F est un « *flip set* » et donc *I* remporte la partie supérieure, contredisant ainsi le fait que *II* possède une stratégie gagnante.

On montre de manière tout à fait analogue que si le premier joueur possède une stratégie gagnante, alors le second en possède une également.

□ *Prop. 3.3.4*

3.3.2 La classe des ensembles boréliens

Si l'Axiome du Choix permet d'« exhiber » des jeux non déterminés, il existe *a contrario* une vaste classe d'ensembles pour laquelle tous les jeux de

Gale-Stewart sont déterminés. Sa définition est topologique : c'est la classe des ensembles boréliens.

Pour cela, on munit les ensembles de mots infinis A^ω de la topologie usuelle : un *ouvert de base* est un sous-ensemble quelconque de la forme $N_u = uA^\omega$ pour $u \in A^+$. Un *ouvert* est alors défini comme un ensemble de la forme $\bigcup_{u \in U} N_u$ pour un ensemble $U \subseteq A^*$ quelconque [17].

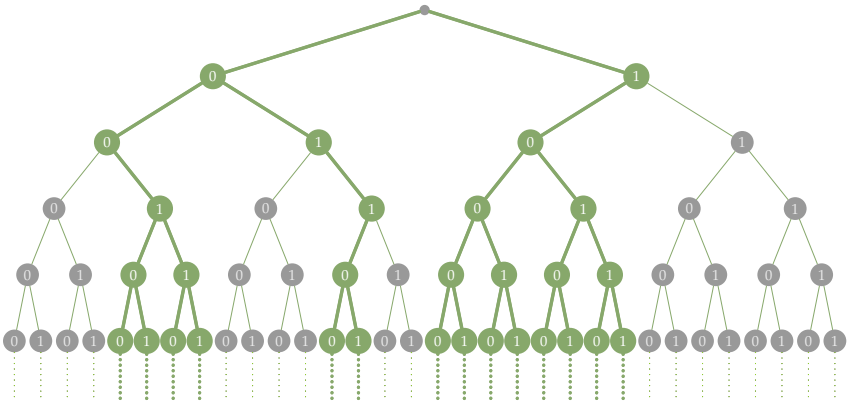


FIGURE 3.9 – *Un ouvert de l'espace* $\{0,1\}^\omega$.

Définition 3.3.5. *La classe des sous-ensembles* boréliens *de* A^ω *est la plus petite classe qui contienne les ouverts et soit close par les opérations de réunion dénombrable et de complémentation.*

Exemple 3.3.6. *Les sous-ensembles de* $\{0,1\}^\omega$ *qui suivent sont boréliens.*

1. $\{0^\omega\}$ *: c'est un ensemble fermé (le complémentaire d'un ensemble ouvert) puisque* $\{0^\omega\}^\complement = \bigcup_{u \in \{0,1\}^* 1} N_u$ *;*
2. $\{0,1\}^* 1^\omega$ *: car il s'écrit* $\bigcup_{u \in \{0,1\}^* 1} u1^\omega$ *et forme ainsi une réunion dénombrable d'ensembles fermés.*

Pour un arbre $T \subseteq A^*$, on note $[T]$ l'ensemble des ses branches infinies [18]. Les sous-ensembles de A^ω de la forme $[T]$ pour un arbre $T \subseteq A^*$ sont en fait précisément les ensembles *fermés* de cet espace [67].

17. A noter que les cas $U = \emptyset$ et $U = A$ engendrent respectivement $\emptyset$ et A^ω. A noter également que muni de cette topologie, l'espace des mots infinis d'entiers $\mathbb{N}^\omega$ est très proche de celui des réels muni de la topologie usuelle (celle dans laquelle les ouverts de base sont de la forme $]x, y[$) puisqu'il est homéomorphe — c'est-à-dire isomorphe pour la structure topologique — à $\mathbb{R} \setminus \mathbb{Q}$.

18. $[T] = \{\vec{a} \in A^\omega \mid \forall n \in \mathbb{N}\, \vec{a} \restriction n \in T\}$ et $[T]^\complement = \bigcup_{v \notin T} vA^\omega$.

Dès leur introduction, les ensembles boréliens ont été classifiés en une hiérarchie dans laquelle la complexité d'un ensemble est fonction du nombre minimal d'opérations de réunion ou d'intersection dénombrables nécessaires pour le produire sur la base d'ensembles ouverts ou fermés.

Définition 3.3.7 (Hiérarchie borélienne)**.** *Par induction sur les ordinaux, on définit*

$$\mathbf{\Sigma}^0_1 = \{\textit{ouverts}\}; \quad \mathbf{\Pi}^0_\alpha = \{E^{\complement} \mid E \in \mathbf{\Sigma}^0_\alpha\}; \quad \mathbf{\Sigma}^0_\alpha = \left\{ \bigcup_{n\in\mathbb{N}} E_n \mid E_n \in \bigcup_{\beta<\alpha} \mathbf{\Pi}^0_\beta \right\}.$$

On utilise également la notation $\mathbf{\Delta}^0_\alpha = \mathbf{\Sigma}^0_\alpha \cap \mathbf{\Pi}^0_\alpha$.

Ainsi, parmi les sous-ensembles de l'espace $\{0,1\}^\omega$, on a : $\{0^\omega\} \in \mathbf{\Pi}^0_1$; $\{0,1\}^*1^\omega \in \mathbf{\Sigma}^0_2$; $(\{0,1\}^*0)^\omega \in \mathbf{\Pi}^0_2$. En effet, $\{0,1\}^\omega = [\{0,1\}^*]$, $\{0,1\}^*1^{\omega\complement} = (\{0,1\}^*0)^\omega$ et $(\{0,1\}^*0)^\omega = \underbrace{\bigcap_{n\in\mathbb{N}} \underbrace{(\{0,1\}^*0)^n \{0,1\}^\omega}_{\mathbf{\Sigma}^0_2 1}}_{\mathbf{\Pi}^0_2}$.

Exemple 3.3.8. *Soit* $\mathscr{A} = (A, Q, q_i, \delta, F)$ *un automate de Büchi* déterministe *et* $\vec{a}$ *un mot infini, on a*

$$\begin{aligned} \vec{a} \in \mathscr{L}(\mathscr{A}) &\iff \textit{il existe une infinité d'entiers } n \textit{ tels que } \rho_{\vec{a}}(n) \in F \\ &\iff \forall m\, \exists n > m, \rho_{\vec{a}}(n) \in F \\ &\iff \forall m\, \exists n > m, \rho_{\vec{a}} \in \mathscr{N}_n \end{aligned}$$

où $\mathscr{N}_n$ *désigne* $\{\rho \in \{Q\}^\omega \mid \rho(n) \in F\}$ *qui est un ensemble ouvert. On remarque très facilement que la fonction* f *qui associe à tout mot infini* $\vec{a} \in A^\omega$ *son exécution* $\rho_{\vec{a}} \in Q^\omega$ *est continue*[19]*. Par conséquent* $\mathscr{O}_n = \{\vec{a} \in \{A\}^\omega \mid \rho_{\vec{a}} \in \mathscr{N}_n\} = f^{-1}\mathscr{N}_n$ *est également ouvert. D'où*

$$\mathscr{L}(\mathscr{A}) = \underbrace{\bigcap_{m\in\mathbb{N}} \overbrace{\bigcup_{n>m}}^{\mathbf{\Sigma}^0_1} \mathscr{O}_n}_{\mathbf{\Pi}^0_2}.$$

Dès que l'alphabet A possède au moins deux lettres, cette hiérarchie — qui possède autant de niveaux qu'il y a d'ordinaux dénombrables[20] — est formée d'inclusions strictes schématisées par les flèches de la figure suivante.

19. Cela s'établit, entre autres, au moyen du jeu présenté dans la Définition 3.3.15.

20. il faut voir un ordinal dénombrable soit comme un entier, soit comme une manière particulière d'ordonner l'ensemble des entiers en sorte que l'ordre soit à la fois total et respecte le fait que tout sous-ensemble non vide possède un plus petit élément.

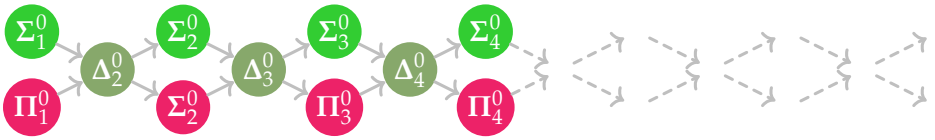


FIGURE 3.10 – *Hiérarchie borélienne.*

Cette hiérarchie caractérise en quelque sorte les ensembles boréliens par le bas. Une autre manière de les caractériser, cette fois par le haut, repose sur un résultat de Souslin [67].

Définition 3.3.9. *Un ensemble $\mathscr{A} \subseteq A^\omega$ est* analytique *s'il existe un arbre $T \subseteq (\mathbb{N} \times A)^*$ tel que*

$$\vec{a} \in \mathscr{A} \iff \exists \vec{x} \in \mathbb{N}^\omega, \vec{x} \times \vec{a} \in [T]$$

où $\vec{x} \times \vec{a}$ représente le mot infini $(x_0, a_0)(x_1, a_1)(x_2, a_2) \cdots$.

Il est à noter que si l'on remplace l'ensemble fermé $[T]$ par un ensemble borélien quelconque on obtient toujours également un ensemble analytique.

Théorème 3.3.10 (Souslin)**.** *Pour tous ensembles A dénombrable et $B \subseteq A^\omega$,*

$$B \text{ est borélien} \iff B \text{ et } B^\complement \text{ sont tous deux analytiques.}$$

Exemple 3.3.11. *Soit $\mathscr{A} = (A, Q, q_i, \Delta, F)$ un automate de Büchi* non déterministe *et $\vec{a}$ un mot infini, on a*

$$\begin{aligned} \vec{a} \in \mathscr{L}(\mathscr{A}) &\iff \text{il existe } \rho_{\vec{a}} \text{ et une infinité d'entiers } n \text{ t.q. } \rho_{\vec{a}}(n) \in F \\ &\iff \exists \rho_{\vec{a}} \, \forall m \, \exists n > m, \rho_{\vec{a}}(n) \in F \\ &\iff \exists \rho \, (\rho, \vec{a}) \in \underbrace{\bigcap_m G_m}_{\mathbf{\Pi}^0_2} \end{aligned}$$

où G_m désigne le sous-ensemble suivant qui est un ouvert *:*

$$G_m = \{(\rho, \vec{a}) \in \mathbb{N}^\omega \times A^\omega \mid (\rho_m, a_m, \rho_{m+1}) \in \Delta \wedge \exists n > m \, \rho_n \in F\}.$$

Ce langage est la projection d'un ensemble borélien, il est donc analytique. Mais puisque les langages ω-réguliers sont clos par complémentation, on en déduit que $\mathscr{L}(\mathscr{A})$ est borélien.

L'intérêt de la classe des boréliens pour les jeux qui nous occupent réside dans le résultat qui suit.

Théorème 3.3.12 (Martin). *Soit A un ensemble et $B \subseteq A^\omega$ borélien, le jeu de Gale-Stewart $\mathcal{G}(B)$ est déterminé.*

Bien qu'elle ne requiert que des notions basiques de topologie et d'arithmétique ordinale, la preuve de ce résultat [85] est difficile. Le lecteur curieux pourra consulter [67].

3.3.3 La réduction continue

Définition 3.3.13. *Une fonction $f : A^\omega \to B^\omega$ est* continue *si pour tout ouvert $\mathcal{O} \subseteq B^\omega$, $f^{-1}\mathcal{O}$ est un ouvert de A^ω.*

La notion de continuité est centrale en topologie. Elle est généralement introduite sur les réels en disant qu'une fonction continue est telle que l'on peut tracer sa courbe « sans lever le crayon ». Ici, pour les espaces de la forme A^ω, il n'est pas question de courbe. Pourtant il existe une caractérisation tout aussi jolie, à l'aide de jeux infinis.

Définition 3.3.14. *Soit $f : A^\omega \to B^\omega$, le jeu caractérisant les fonctions continues $\mathscr{C}(f)$ est un jeu infini dans lequel les joueurs (I et II) choisissent à tour de rôle $a \in A$ et $b \in B$. Le joueur I débute la partie. Le joueur II peut passer son tour (contrairement au joueur I) et il emporte la partie si la suite $\vec{b}$ qu'il a joué vérifie*[21] *$f(\vec{a}) = \vec{b}$. Dans le cas contraire, le joueur I gagne.*

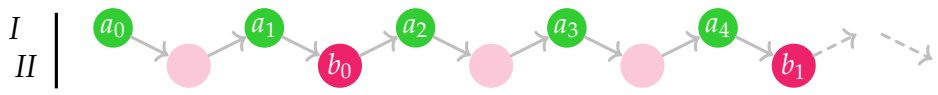


FIGURE 3.11 – *Jeu caractérisant les fonctions continues.*

Proposition 3.3.15. *Soit $f : A^\omega \to B^\omega$, le joueur II possède une stratégie gagnante dans $\mathscr{C}(f)$ $\iff$ f est continue.*

Preuve de la Proposition 3.3.15. Un simple exercice ludique. □ *Prop. 3.3.15*

Dans le domaine de la complexité algorithmique, on trouve les notions de problèmes **Γ**-difficile et **Γ**-complet. Toutes deux reposent sur une relation de réduction basée généralement sur celle de fonction calculable en temps polynomial ou bien en espace logarithmique. De la sorte, un problème P est plus simple (au sens large) qu'un problème P' s'il existe une fonction « *simple* » f telle que la question de savoir si x satisfait P se ramène à celle de savoir si $f(x)$ satisfait P'. En topologie, la simplicité d'une fonction renvoie

21. En particulier $\vec{b}$ est infinie.

généralement à son caractère continu. Cela induit la notion suivante de réduction continue dénommée « *réduction de Wadge* ».

Définition 3.3.16. *Soient $L \subseteq A^\omega$ et $M \subseteq B^\omega$,*

$$L \leqslant_w M \iff \text{il existe } f : A^\omega \xrightarrow{\text{continue}} B^\omega \text{ telle que } L = f^{-1}M.$$

On note $L <_w M$ lorsque $L \leqslant_w M$ et $M \not\leqslant_w L$. On écrit également $L \equiv_w M$ pour $L \leqslant_w M$ et $M \leqslant_w L$.

La notion de fonction continue se laissant caractériser à l'aide d'un jeu infini, celle de réduction continue s'exprime tout naturellement sous forme de l'existence d'une stratégie gagnante dans un jeu (dénommé « *jeu de Wadge* »).

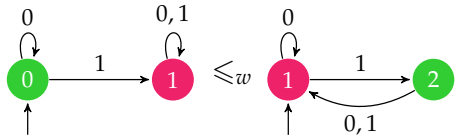
Définition 3.3.17. *Soient $L \subseteq A^\omega$ et $M \subseteq B^\omega$, les règles du jeu $\mathscr{W}(L, M)$ sont identiques au jeu $\mathscr{C}(f)$ de la Définition 3.3.14, la condition de gain devenant II gagne si $\vec{a} \in L \iff \vec{b} \in M$.*

Remarque 3.3.18. *Il est assez facile de construire à partir d'ensembles boréliens quelconques $L \subseteq A^\omega$ et $M \subseteq B^\omega$ un ensemble $N \subseteq (A \cup B)^\omega$ en sorte qu'un joueur possède une stratégie gagnante dans le jeu $\mathscr{G}(N)$ si et seulement si ce* même joueur *possède une stratégie gagnante dans le jeu $\mathscr{W}(L, M)$. Autrement dit, par le Théorème 3.3.12, les jeux de Wadge impliquant des ensembles boréliens sont tous déterminés.*

Nous nous permettons désormais l'abus de langage qui consiste à mentionner des automates de parité en lieu et place des langages qu'ils reconnaissent.

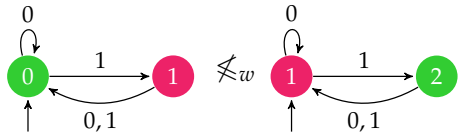
Exemple 3.3.19. *La relation ci-dessous est vérifiée puisque la stratégie suivante est gagnante pour le joueur II dans le jeu de Wadge sous-jacent :*

— *tant que I ne joue que des « 0 », II ne joue que des « 1 » ;*
— *si I joue un « 1 », II ne joue que des « 0 » jusqu'à la fin de la partie.*



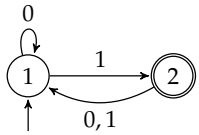
Exemple 3.3.20. *La relation ci-dessous est vérifiée car le joueur I possède la stratégie gagnante suivante :*

— *I copie le jeu de II, tout en jouant « 0 » chaque fois que II passe son tour.*



Un langage est *complet* pour une classe topologique s'il en fait partie et réduit tous les langages de cette classe.

Exemple 3.3.21. *L'automate de Büchi $\mathscr{B}$ ci-dessous est $\mathbf{\Pi}^0_2$-complet.*



En effet, en tant qu'automate déterministe de Büchi, $\mathscr{L}(\mathscr{B}) \in \mathbf{\Pi}^0_2$ (voir Exemple 3.3.8). Par ailleurs, soit $B = \bigcap_{n\in\mathbb{N}} \mathscr{O}_n$ un ensemble $\mathbf{\Pi}^0_2$ quelconque[22] *de A^ω. La stratégie suivante est gagnante pour le joueur II dans le jeu $\mathscr{W}(B, \mathscr{L}(\mathscr{B}))$:*

— *II utilise un compteur initialisé à $c := 0$ et répond à la position $a_0 \cdots a_n$ de I par*
  — *« 0 » si $a_0 \cdots a_n A^\omega \not\subseteq \mathscr{O}_c$;*
  — *« 1 » si $a_0 \cdots a_n A^\omega \subseteq \mathscr{O}_c$, puis incrémente $c := c + 1$.*

La relation de réduction continue $\leqslant_w$ constitue un relation d'ordre partielle : elle est réflexive[23] et transitive[24]. Si l'on se restreint à une classe de langages pour laquelle les jeux de Wadge sont déterminés (p. ex. les ensembles boréliens), cette relation acquiert alors des propriétés fascinantes :

1. les anti-chaînes ont une longueur au plus deux : il n'est pas possible de trouver trois ensembles différents qui soient incomparables deux à deux ;
2. l'ordre est bien-fondé : Il n'existe pas de suite infinie d'ensembles vérifiant $E_0 >_w E_1 >_w E_2 >_w \cdots$.

Pour la seconde propriété voir [67]. La première est une conséquence immédiate de

Lemme 3.3.22 (Wadge). *Soient $L \subseteq A^\omega$ et $M \subseteq B^\omega$. Si $\mathscr{W}(L, M)$ est déterminé, alors*

$$L \not\leqslant_w M \Rightarrow M \leqslant_w L^{\complement}.$$

22. Chaque $\mathscr{O}_n$ étant ouvert.
23. L'identité est une fonction continue.
24. La classe des fonctions continues est close par composition.

Preuve du Lemme 3.3.22. Le jeu $\mathscr{W}(L, M)$ étant déterminé, si $L \not\leqslant_w M$, alors le joueur I possède une stratégie gagnante dans $\mathscr{W}(L, M)$ qui induit immédiatement une stratégie gagnante pour II dans $\mathscr{W}\left(M, L^{\complement}\right)$. □ *Lem. 3.3.22*

Une *classe de Wadge* est une classe topologique[25] qui possède un ensemble complet pour cette classe. Ainsi, si E et E' sont respectivement complets pour les classes $\mathbf{\Gamma}$ et $\mathbf{\Gamma}'$, alors $E \leqslant_w E' \iff \mathbf{\Gamma} \subseteq \mathbf{\Gamma}'$. Lorsqu'on ordonne les classes de Wadge par inclusion, on parle alors de *hiérarchie de Wadge*. Restreinte aux ensembles boréliens, celle-ci a la forme ci-dessous. Elle ressemble en tous points à la hiérarchie borélienne, à la seule différence

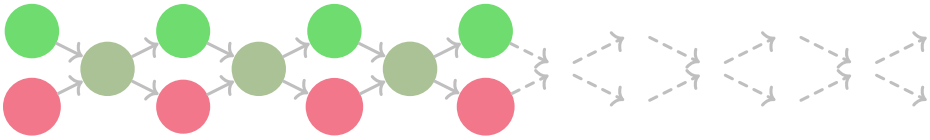


FIGURE 3.12 – *Hiérarchie de Wadge.*

qu'elle est beaucoup plus fine. En effet, le second niveau de la hiérarchie borélienne correspond à un niveau non dénombrable !

Proposition 3.3.23. *Si $\mathscr{A}$ est un automate de parité déterministe, alors*

$$\mathscr{L}(\mathscr{A}) \in \mathbf{\Delta}^0_3 = \mathbf{\Sigma}^0_3 \cap \mathbf{\Pi}^0_3.$$

Preuve de la Proposition 3.3.23. On a $\vec{a} \in \mathscr{L}(\mathscr{A})$

$$\begin{aligned} &\iff \exists i \leqslant p \left(\exists^\infty n \, \mathcal{C}\left(\rho_{\vec{a}}(n)\right) = 2i \wedge \exists m \, \forall n \geqslant m \, \mathcal{C}\left(\rho_{\vec{a}}(n)\right) \leqslant 2i \right) \\ &\iff \underbrace{\exists i \leqslant p \left(\underbrace{\forall m \, \exists n > m \, \mathcal{C}\left(\rho_{\vec{a}}(n)\right) = 2i}_{\mathbf{\Pi}^0_2} \wedge \underbrace{\exists m \, \forall n \geqslant m \, \mathcal{C}\left(\rho_{\vec{a}}(n)\right) \leqslant 2i}_{\mathbf{\Sigma}^0_2} \right)}_{\mathbf{\Sigma}^0_3}. \end{aligned}$$

Comme $\mathscr{L}(\mathscr{A})^{\complement} = \mathscr{L}(\mathscr{A}^{\complement})$, on obtient également $\mathscr{L}(\mathscr{A}) \in \mathbf{\Pi}^0_3$. □ *Prop. 3.3.23*

Les langages ω-réguliers sont donc situés dans les trois premiers niveaux de la hiérarchie borélienne. On trouve des langages complets pour les classes $\mathbf{\Sigma}^0_1$, $\mathbf{\Pi}^0_1$, $\mathbf{\Sigma}^0_2$, $\mathbf{\Pi}^0_2$. Par contre, il n'en est pas de $\mathbf{\Sigma}^0_3$-complet. En jouant les jeux de Wadge sous-jacents, on se convainc très facilement du caractère complet des langages proposés dans la figure 3.13.

Toutefois, comme l'ont montré Wagner [133] et Selivanov [121], l'ordre de Wadge permet une analyse beaucoup plus fine. Pour cela, considérons

25. Une classe de sous-espaces topologiques close par image inverse de fonctions continues.

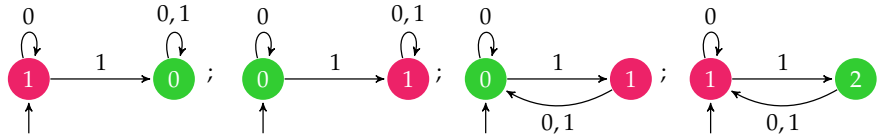


FIGURE 3.13 – *Langages ω-réguliers complets pour les classes $\mathbf{\Sigma}^0_1$, $\mathbf{\Pi}^0_1$, $\mathbf{\Sigma}^0_2$, $\mathbf{\Pi}^0_2$.*

tout d'abord l'ensemble des suites finies décroissantes (au sens large) d'entiers naturels, muni de l'ordre lexicographique $\leqslant_{lex}$. C'est un bon ordre [26] : un ordre total dont tout sous-ensemble non vide possède un élément minimal.

A chaque suite finie décroissante non vide d'entiers nous allons associer un automate de parité déterministe $\mathscr{A}_u$ en sorte que si u et v sont deux telles suites, alors l'ordre lexicographique entre elles et l'ordre de Wadge entre automates coïncident : $u <_{lex} v \iff \mathscr{A}_u <_w \mathscr{A}_v$.

En premier lieu, définissons pour chaque entier n un automate déterministe de parité $\mathscr{A}_n$. Les cas $n = 0$ et n successeurs sont décrits dans les figures qui suivent.

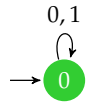


FIGURE 3.14 – *Automate $\mathscr{A}_0$.*

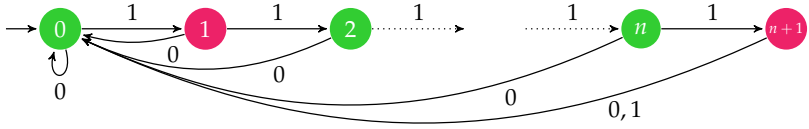


FIGURE 3.15 – *Automate $\mathscr{A}_{n+1}$ (les couleurs correspondent à n pair).*

Ensuite, à toute suite $u = n_k n_{k-1} \cdots n_0$ vérifiant $n_k \geqslant n_{k-1} \geqslant \cdots \geqslant n_0$, nous associons trois automates de parité $\mathscr{A}_u$, $-\mathscr{A}_u$, et $\pm\mathscr{A}_u$ dont les graphes sont représentés par les trois figures qui suivent. L'étiquetage est quelconque *pour autant qu'il rende ces automates déterministes*.

Exemple 3.3.24. *Un étiquetage déterministe pour l'automate $\mathscr{A}_{7100}$.*

26. Ce bon ordre est isomorphe à l'ordinal ω^ω. L'isomorphisme faisant correspondre à la suite $n_k \geqslant n_{k-1} \geqslant \cdots \geqslant n_0$ l'ordinal $\omega^{n_k} + \omega^{n_{k-1}} + \cdots + \omega^{n_0}$.

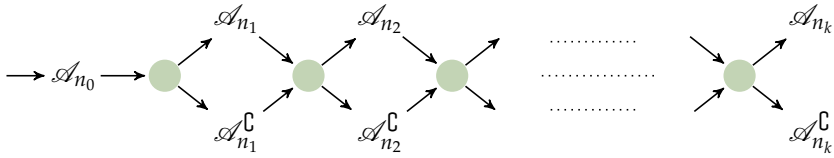


FIGURE 3.16 – *Automate* $\mathscr{A}_{n_k \cdots n_0}$.

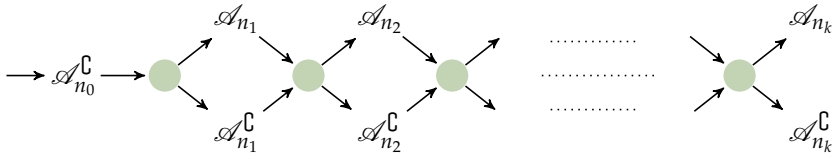


FIGURE 3.17 – *Automate* $-\mathscr{A}_{n_k \cdots n_0}$.

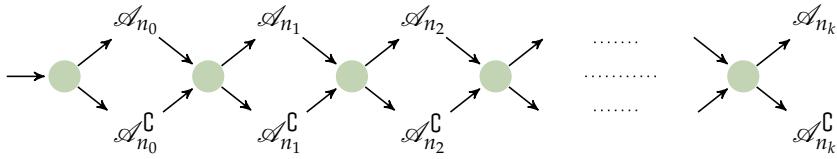


FIGURE 3.18 – *Automate* $\pm\mathscr{A}_{n_k \cdots n_0}$.

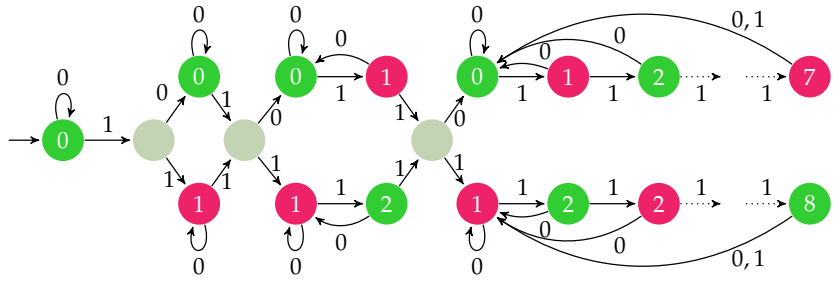


FIGURE 3.19 – *Automate* $\mathscr{A}_{7100}$.

Proposition 3.3.25. *Soient u, v deux suites finies non vides décroissantes d'entiers. Alors,*

1. $\mathscr{A}_u \not\leqslant_w -\mathscr{A}_u$ *et* $-\mathscr{A}_u \not\leqslant_w \mathscr{A}_u$;
2. $\mathscr{A}_u <_w \pm\mathscr{A}_u$ *et* $-\mathscr{A}_u <_w \pm\mathscr{A}_u$;
3. *si* $u <_{lex} v$, *alors* $\pm\mathscr{A}_u <_w \mathscr{A}_v$ *et* $\pm\mathscr{A}_u <_w -\mathscr{A}_v$.

Preuve de la Proposition 3.3.25. Les preuves consistent, pour chaque jeu de réduction sous-jacent, à mettre à jour le joueur possédant une stratégie gagnante. C'est un procédé aussi facile que fastidieux. La preuve du

cas 3 nécessite d'utiliser l'induction le long de l'ordre lexicographique. □ *Prop. 3.3.25*

Théorème 3.3.26. *Si $\mathscr{A}$ est un automate déterministe de parité, alors il existe une suite finie décroissante non vide d'entiers u telle que l'une seule des trois possibilités suivante est vérifiée :*

$$\mathscr{A} \equiv_w \mathscr{A}_u; \qquad \mathscr{A} \equiv_w -\mathscr{A}_u; \qquad \mathscr{A} \equiv_w \pm\mathscr{A}_u.$$

Preuve du Théorème 3.3.26. Les preuves reposent sur l'étude des composantes strictement connexes du graphe de $\mathscr{A}$ et de leurs relations d'accessibilité. □ *Thm. 3.3.26*

Automates à compteur et automates d'arbres

Si l'on substitue à la notion d'automate celle d'une machine plus puissante, le problème de l'étude de la complexité topologique des langages reconnus devient très vite inextricable. Ainsi, comme l'a montré Finkel, les langages de mots infinis reconnus par automates de Büchi à un compteur sont exactement de la même complexité topologique que ceux reconnus par machines de Turing avec condition de Büchi [34].

Une autre démarche consiste à conserver la notion d'automate et remplacer les mots infinis par des arbres infinis. En le lisant de gauche à droite, étage par étage, un arbre infini (sur un alphabet fini) peut être regardé comme un mot infini. L'étude de la complexité topologique des langages ω-réguliers s'étend donc tout naturellement aux langages d'arbres.

Dans le cadre des automates de parité déterministes, Niwiński et Walukiewicz [103] ont montré que les langages reconnus sont soit complets pour la classe des co-analytiques, soit résident à l'intérieur de la classe $\mathbf{\Pi}^0_3$. Ultérieurement, Murlak a entièrement déterminé quelles sont les classes de Wadge incriminées [27] [96]. Toutefois, dans le cas des automates d'arbres non-déterministes, l'étude des langages ω-réguliers demeure encore, sous bien des aspects, un mystère à percer.

27. C'est un résultat beaucoup plus difficile que celui sur les automates de mots infinis que nous avons esquissé ici.

Chapitre 4

Une introduction à la réalisabilité classique

4.1 Introduction

Depuis une quarantaine d'années, la compréhension de la logique mathématique a été profondément renouvelée par la mise en évidence d'une correspondance intime entre les concepts de la *théorie de la démonstration* et ceux de la *programmation fonctionnelle*. Selon cette correspondance, dite de Curry-Howard, toute *formule* A correspond à un *type*, et toute *preuve*[1] de A correspond à un *programme* du type correspondant. Ainsi, là où le logicien voyait la règle du *modus ponens* comme un moyen de combiner une preuve de $A \Rightarrow B$ avec une preuve de A pour construire une preuve de B, l'informaticien voit à présent dans cette règle l'expression du fait que tout programme de type $A \Rightarrow B$ peut être *appliqué* à un programme de type A pour donner un programme de type B.

La découverte de cette correspondance entre les preuves et les programmes a eu des répercussions considérables à la fois sur le plan théorique et sur le plan pratique. Elle a permis notamment l'émergence de toute une famille d'assistants à la démonstration mathématique entièrement basés sur ce principe, dont l'un des plus connus est l'assistant Coq [127].

Pendant longtemps, la correspondance preuves/programmes a été limitée à la logique *intuitionniste* (ou *constructive*), c'est-à-dire à la logique classique privée de la loi du tiers-exclu $A \vee \neg A$ et du principe de raisonnement par l'absurde (ces deux principes étant équivalents). Et ce n'est que dans les années 1990 (notamment suite aux travaux de Griffin [39]) qu'on a commencé à comprendre qu'il était possible d'étendre l'interprétation

1. On utilisera ici les mots *preuve* et *démonstration* comme des synonymes.

calculatoire des preuves à toute la logique classique, à condition d'autoriser les programmes à commettre des erreurs, quitte à revenir ensuite en arrière dans leur exécution (par un mécanisme de *backtrack* sur lequel nous aurons l'occasion de revenir en détail par la suite).

C'est précisément pour comprendre — et prédire — le comportement subtil des programmes issus des preuves classiques (qui, contrairement à leurs homologues intuitionnistes, se permettent de « bluffer » quitte à se dédire plus loin) que Jean-Louis Krivine a introduit la théorie de la *réalisabilité classique* [71] dont on se propose de présenter les bases ici.

Pour l'informaticien, la réalisabilité classique apparaîtra donc comme une méthode sémantique permettant d'analyser finement le comportement des programmes extraits à partir des preuves en logique classique. Tandis que pour le logicien, elle apparaîtra comme une reformulation audacieuse de la théorie des modèles à l'aide des outils de la sémantique opérationnelle et de la programmation fonctionnelle.

Plan du chapitre Bien que les travaux les plus récents dans ce domaine soulèvent des perspectives prometteuses en théorie des modèles [2], nous nous concentrerons davantage sur les aspects liés à la preuve et à la programmation dans le cadre de cette introduction, dont le but est d'initier le lecteur à l'extraction de programme en logique classique.

Pour cela, nous allons nous placer dans l'arithmétique classique du second ordre (Section 4.2), dont nous rappellerons le langage, les règles de déduction et les axiomes. Nous montrerons ensuite comment associer à chaque démonstration un programme (Section 4.3) écrit dans un langage fonctionnel permettant de sauvegarder et restaurer à tout moment le contexte d'évaluation (mécanisme de *backtrack*).

La construction d'un *modèle de réalisabilité classique* (Section 4.4) nous permettra ensuite d'associer à chaque formule du langage un ensemble de programmes partageant un comportement calculatoire commun, avant de démontrer (dans un *théorème d'adéquation*) que tous les programmes extraits à partir des preuves classiques ont le comportement attendu. Enfin, nous verrons (Section 4.5) comment la combinaison de ces techniques permet d'extraire à partir de toute démonstration — même non constructive — d'une formule de la forme $(\exists x \in \mathbf{N})\, f(x) = 0$ un *témoin existentiel*, c'est-à-dire un entier n satisfaisant l'équation $f(n) = 0$.

2. Notamment en lien avec le *forcing* de Cohen, dont la réalisabilité classique apparaît de plus en plus comme une généralisation, voir à ce sujet [72, 88, 73]. La réalisabilité classique permet déjà d'établir certains résultats d'indépendance (pour l'instant assez pathologiques) sur les cardinaux des sous-ensembles de $\mathbb{R}$ en théorie des ensembles, voir [73].

4.2 Logique et arithmétique du second ordre

4.2.1 Présentation

La logique du second ordre est une extension de la logique du premier ordre [3] dans laquelle on distingue deux types d'objets :

— les *individus* (ou *objets du premier ordre*), qui sont les objets de base de la théorie considérée ;

— les *relations* sur les individus (ou *objets du second ordre*), d'arité quelconque (représentée par un entier $k \geqslant 0$).

Dans ce qui suit, on considérera uniquement le cas de l'*arithmétique*, dans laquelle les individus sont les entiers naturels [4].

Contrairement à la logique du premier ordre, la logique du second ordre traite les relations (sur les individus) comme des citoyens de première classe, en introduisant un jeu de variables spécifique pour les représenter, ainsi que les quantifications correspondantes.

Formellement, on travaille donc avec deux jeux de variables :

— des variables du premier ordre, notées x, y, z, etc. (en nombre infini dénombrable) destinées à représenter des *entiers naturels* ;

— des variables du second ordre, notées X, Y, Z, etc. (en nombre infini dénombrable) destinées à représenter des *relations* sur les entiers. Comme il est possible de considérer des relations d'arité 1, 2, 3, etc. (et même d'arité 0), on suppose que ce jeu de variables est lui-même scindé en une infinité de sous-ensembles (infinis dénombrables), un pour chaque arité $k \geqslant 0$. Dans ce qui suit, on introduira librement les variables du second ordre sans préciser leur arité, celle-ci pouvant être aisément inférée à partir du contexte.

4.2.2 Les termes du premier ordre

Comme en logique du premier ordre, les *termes du premier ordre* sont construits à partir des variables du premier ordre à l'aide d'un certain nombre de *symboles de fonction* (notation : f, g, h, etc.) munis d'une arité. Comme d'habitude, on appelle *symbole de constante* tout symbole de fonction d'arité 0, et on suppose que les symboles de fonction du langage sont déclarés (avec leurs arités) dans une *signature du premier ordre*, notée Σ.

Formellement, les *termes du premier ordre* (notation : e, e', etc.) sont

3. Dans ce qui suit, on suppose connues les bases de la *logique du premier ordre* (qu'on appelle également le calcul des prédicats du premier ordre), voir [28] par exemple.

4. Cette affirmation sera relativisée (sans jeu de mots) à la Section 4.2.6.

définis comme en logique du premier ordre à partir de la grammaire :

Termes du premier ordre $\quad e, e_1, \ldots \quad ::= \quad x \quad | \quad f(e_1, \ldots, e_k)$

où x parcourt l'ensemble des variables du premier ordre, et f l'ensemble des symboles de fonction d'arité k (quelconque) dans la signature Σ.

Comme d'habitude, on note $FV(e)$ l'ensemble (fini) des variables libres d'un terme e, et la substitution correspondante est notée $e\{x := e'\}$. (On rappelle que cette notation désigne le terme obtenu en remplaçant dans e toutes les occurrences de la variable x par le terme e'.)

Dans le cadre de l'arithmétique, on fait en outre les hypothèses suivantes sur les symboles définissant le langage.

1. La signature Σ comporte un symbole de constante 0 (zéro) et un symbole de fonction (d'arité $k \geqslant 1$) pour chaque fonction récursive primitive (à k arguments). Dans ce qui suit, on utilisera les symboles de fonction s (successeur), $+$ (addition), $-$ (soustraction tronquée [5]) et $\times$ (multiplication).
2. Chaque symbole de fonction f d'arité k (attaché à une fonction récursive primitive particulière) est interprété dans $\mathbb{N}$ par la fonction récursive primitive correspondante, notée $f^{\mathbb{N}} : \mathbb{N}^k \to \mathbb{N}$. Le symbole de constante 0 est interprété par l'entier $0^{\mathbb{N}} = 0$.

Dans la mesure où les termes du premier ordre vont nous servir à représenter des entiers naturels, on les appellera également des *expressions arithmétiques*, afin notamment de ne pas les confondre avec les termes du λ_c-calcul que nous définirons à la Section 4.3. Comme d'habitude, la valeur d'une expression arithmétique close e (c'est-à-dire sa dénotation dans $\mathbb{N}$) est notée $e^{\mathbb{N}}$, et est définie par induction sur e en interprétant le symbole 0 par 0 et chaque symbole de fonction f par la fonction $f^{\mathbb{N}}$ correspondante.

4.2.3 Les formules du second ordre

Le langage des *formules* (notation : A, B, C, etc.) de la logique minimale du second ordre est défini à partir de la grammaire suivante :

Formules $\quad A, B \quad ::= \quad X(e_1, \ldots, e_k) \quad | \quad A \Rightarrow B \quad | \quad \forall x\, A \quad | \quad \forall X\, A$

où X parcourt l'ensemble des variables du second ordre d'arité $k \geqslant 0$ (quelconque), et $e_1, \ldots, e_k$ l'ensemble des expressions arithmétiques.

Le langage ci-dessus comporte un certain nombre de différences par rapport au langage usuel de la logique du premier ordre. Par exemple :

5. On rappelle que la *soustraction tronquée* est définie dans $\mathbb{N}$ par les équations $x - 0 = x$, $0 - x = 0$ et $s(x) - s(y) = x - y$.

1. Les variables du second ordre (X, Y, Z, etc.) s'utilisent ici comme des symboles de prédicat, et la formule atomique $X(e_1, \ldots, e_k)$ signifie : « le k-uplet d'entiers $e_1, \ldots, e_k$ satisfait la relation X ». Dans le langage présenté ici, les variables du second ordre sont les seuls symboles de prédicat autorisés.
2. Le langage logique est très pauvre, puisqu'il ne comporte que l'implication et la quantification universelle — d'où la terminologie de logique *minimale* du second ordre. Nous verrons cependant que ce langage est suffisant pour exprimer toutes les constructions habituelles de la logique, à l'aide de codages appropriés.
3. En logique du second ordre, il convient de distinguer deux types de quantification universelle : la quantification universelle du premier ordre $\forall x\, A$, qui porte sur les individus, et la quantification universelle du second ordre $\forall X\, A$, qui porte sur les relations d'arité $k \geqslant 0$ (celle-ci étant implicitement donnée par la variable X).

Les notions de *variable libre* et de *variable liée* sont définies comme en logique du premier ordre, les deux types de variables se comportant de la même manière à cet égard. En particulier : toutes les occurrences libres de la variable x (resp. de la variable X) dans la formule A sont des occurrences liées dans la formule $\forall x\, A$ (resp. $\forall X\, A$). L'ensemble des variables libres d'une formule A est noté $FV(A)$ (il s'agit d'un ensemble fini combinant des variables du premier et du second ordre).

Comme d'habitude, on identifiera les formules α-convertibles, c'est-à-dire les formules qui ne diffèrent que par les noms de variables liées, par exemple : les formules $\forall x\, \forall Y\, Y(x + z)$ et $\forall y\, \forall X\, X(y + z)$. Le passage au second ordre conduit cependant à distinguer deux types de substitution :

1. La substitution du premier ordre, notée $A\{x := e\}$, qui consiste à remplacer dans la formule A toutes les occurrences libres de la variable x (du premier ordre) par l'expression arithmétique e. Cette opération est définie de la manière usuelle, en renommant dans la formule A toutes les variables liées (du premier ordre) susceptibles d'entrer en conflit avec une variable libre de l'expression e.
2. La substitution du second ordre, notée $A\{X := P\}$, qui consiste à remplacer dans la formule A toutes les occurrences libres de la variable X (du second ordre) par une relation concrète, notée P. Il s'agit d'une opération plus délicate, dont nous ne donnerons la définition qu'à la Section 4.2.4.

Codages au second ordre Dans le langage de la logique minimale du second ordre, les autres constructions de la logique sont définies au moyen

des codages suivants [6], dits "intuitionnistes" :

$$\begin{array}{rcll}
\bot & :\equiv & \forall Y\, Y & \text{(Absurdité)} \\
\neg A & :\equiv & A \Rightarrow \bot & \text{(Négation)} \\
A \wedge B & :\equiv & \forall Y\, ((A \Rightarrow B \Rightarrow Y) \Rightarrow Y) & \text{(Conjonction)} \\
A \vee B & :\equiv & \forall Y\, ((A \Rightarrow Y) \Rightarrow (B \Rightarrow Y) \Rightarrow Y) & \text{(Disjonction)} \\
A \Leftrightarrow B & :\equiv & (A \Rightarrow B) \wedge (B \Rightarrow A) & \text{(Équivalence logique)} \\
\exists x\, A(x) & :\equiv & \forall Y\, (\forall x\, (A(x) \Rightarrow Y) \Rightarrow Y) & \text{(Existentielle, 1}^{\text{er}}\text{ ordre)} \\
\exists X\, A(X) & :\equiv & \forall Y\, (\forall X\, (A(X) \Rightarrow Y) \Rightarrow Y) & \text{(Existentielle, 2}^{\text{e}}\text{ ordre)} \\
e_1 = e_2 & :\equiv & \forall Z\, (Z(e_1) \Rightarrow Z(e_2)) & \text{(Égalité de Leibniz)}
\end{array}$$

(où Y et Z sont des variables fraîches du second ordre d'arité 0 et 1, respectivement). Ces codages sont valides en logique intuitionniste comme en logique classique. En logique classique, on peut aussi utiliser les codages "classiques" basés sur les lois de De Morgan, comme par exemple :

$$A \wedge B \;\equiv\; \neg(A \Rightarrow B \Rightarrow \bot), \quad \exists x\, A(x) \equiv \neg\forall x \neg A(x), \quad \text{etc.}$$

4.2.4 Prédicats et substitution du second ordre

En logique, une relation concrète, ou *prédicat*, est simplement une formule abstraite par rapport à certaines de ses variables. Formellement :

Prédicats $\qquad P, Q \quad ::= \quad \widehat{x}_1 \cdots \widehat{x}_k \cdot A_0.$

Ici, le préfixe de variables chapeautées $\widehat{x}_1 \cdots \widehat{x}_k$, qui indique par rapport à quelles variables on abstrait la formule A_0, joue un rôle de lieur similaire à la quantification $\forall x$. En particulier, l'ensemble des variables libres d'un prédicat $P \equiv \widehat{x}_1 \cdots \widehat{x}_k . A_0$ est défini par $FV(P) = FV(A_0) \setminus \{x_1, \ldots, x_k\}$ (les autres variables libres de A_0, du premier ou du second ordre, jouent le rôle de paramètres). Toute variable du second ordre X d'arité k peut être vue comme un prédicat, à savoir le prédicat $\widehat{x}_1 \cdots \widehat{x}_k . X(x_1, \ldots, x_k)$.

L'application d'un prédicat $P \equiv \widehat{x}_1 \cdots \widehat{x}_k . A_0$ à k arguments $e_1, \ldots, e_k$ est définie au moyen de la substitution du premier ordre par

$$P(e_1, \ldots, e_k) \;\equiv\; A_0\{x_1 := e_1, \ldots, x_k := e_k\}.$$

Cette notation permet de définir la *substitution du second ordre* $A\{X := P\}$ (où A est une formule quelconque et P un prédicat de la même arité que la variable X) en remplaçant toutes les sous-formules atomiques de A de

6. Dans ce qui suit, le connecteur $\Rightarrow$ s'écrit associativement à droite, en ce sens que $A \Rightarrow B \Rightarrow C$ doit se lire $A \Rightarrow (B \Rightarrow C)$; c'est-à-dire comme une formule équivalente à $(A \wedge B) \Rightarrow C$. On préférera cependant la forme $A \Rightarrow B \Rightarrow C$, plus proche du calcul.

la forme $X(e_1, \ldots, e_k)$ par la formule $P(e_1, \ldots, e_k)$. Comme pour la substitution du premier ordre, cette opération nécessite de renommer à la volée toutes les variables liées dans A (du premier ou du second ordre) susceptibles d'entrer en conflit avec une des variables libres du prédicat P.

Les prédicats unaires vus comme des ensembles Tout prédicat unaire $P \equiv \widehat{x}.A(x)$ peut être vu comme un ensemble d'individus, à savoir l'ensemble des individus x tels que $A(x)$. Dans ce qui suit, on utilisera les notations suggestives $\{x : A(x)\}$ pour $\widehat{x}.A(x)$ et $e \in P$ pour $P(e)$. Ainsi, si $A(x)$ est une formule[7] dépendant d'une variable x, les quantifications relativisées à un ensemble P (ne dépendant pas de x) sont définies par :

$$(\forall x \in P)\, A(x) :\equiv \forall x\, (P(x) \Rightarrow A(x)),$$
$$(\exists x \in P)\, A(x) :\equiv \forall Y\, (\forall x\, (P(x) \Rightarrow A(x) \Rightarrow Y) \Rightarrow Y).$$

(La quantification existentielle relativisée est implémentée ici comme une variante du codage de $\exists$ présenté à la Section 4.2.3, mais on aurait pu prendre également la formule $\exists x\, (P(x) \wedge A(x))$, qui est équivalente.)

Ces notations permettent en particulier de définir l'ensemble $\mathbf{N}$ des *entiers de Dedekind* (c'est-à-dire le plus petit ensemble contenant 0 et clos par successeur, suivant la définition de Dedekind) par la macro :

$$\begin{aligned}\mathbf{N} &:\equiv \{x \mid \forall Z\, (0 \in Z \Rightarrow \forall y\, (y \in Z \Rightarrow s(y) \in Z) \Rightarrow x \in Z)\},\\ &\equiv \widehat{x}\,.\forall Z\, (Z(0) \Rightarrow \forall y\, (Z(y) \Rightarrow Z(s(y))) \Rightarrow Z(x)).\end{aligned}$$

Nous aurons l'occasion de revenir sur cet ensemble/prédicat par la suite.

Arithmétique du second ordre et analyse La discussion précédente montre que la logique du second ordre n'est au fond qu'une petite théorie des ensembles à deux étages : l'étage des individus, et celui des ensembles d'individus. Comme en arithmétique du second ordre, les individus sont les entiers naturels, les prédicats représentent naturellement les ensembles d'entiers, qu'on peut voir également comme des *nombres réels* à travers un codage approprié. C'est pourquoi l'arithmétique du second ordre est parfois appelée *analyse* par les logiciens[8].

7. Il nous arrivera parfois d'utiliser la notation $A(x)$ pour désigner une formule A dont on désire mettre l'une des variables libres (ici x) en exergue.

8. On rappelle que chez les logiciens, $\mathbb{R} = \mathfrak{P}(\mathbb{N})$. Évidemment, pour un analyste, il y a quand même un peu de travail à faire pour transférer la structure de $\mathbb{R}$ dans $\mathfrak{P}(\mathbb{N})$…

(Axiome)	$\dfrac{}{\Gamma \vdash A}$ si $A \in \Gamma$
($\Rightarrow$-intro/elim)	$\dfrac{\Gamma, A \vdash B}{\Gamma \vdash A \Rightarrow B}$ $\qquad$ $\dfrac{\Gamma \vdash A \Rightarrow B \qquad \Gamma \vdash A}{\Gamma \vdash B}$
($\forall^1$-intro/élim)	$\dfrac{\Gamma \vdash A}{\Gamma \vdash \forall x\, A}$ si $x \notin FV(\Gamma)$ $\qquad$ $\dfrac{\Gamma \vdash \forall x\, A}{\Gamma \vdash A\{x := e\}}$
($\forall^2$-intro/élim)	$\dfrac{\Gamma \vdash A}{\Gamma \vdash \forall X A}$ si $X \notin FV(\Gamma)$ $\qquad$ $\dfrac{\Gamma \vdash \forall X A}{\Gamma \vdash A\{X := P\}}$
(Loi de Peirce)	$\dfrac{}{\Gamma \vdash ((A \Rightarrow B) \Rightarrow A) \Rightarrow A}$

TABLE 4.1 – *Règles de déduction du système* NK2

4.2.5 Le système de déduction

Le langage des termes et des formules étant posé, il s'agit à présent de définir le système de déduction correspondant. On travaille ici en déduction naturelle avec des séquents asymétriques de la forme

Séquents $\Gamma \vdash A$ («Γ thèse A»)

où $\Gamma \equiv A_1, \ldots, A_n$ (l'*antécédent*, ou *contexte*) est une liste finie de formules éventuellement vide, et où A (le *conséquent*) est une formule.

Intuitivement, le séquent $\Gamma \vdash A$ exprime que la formule A est conséquence logique des hypothèses dans Γ. Le séquent sans hypothèse est noté $\vdash A$, et l'ensemble des variables libres d'un contexte $\Gamma \equiv A_1, \ldots, A_n$ est défini par $FV(\Gamma) := FV(A_1) \cup \cdots \cup FV(A_n)$.

Les règles de la *déduction naturelle classique du second ordre* (NK2) sont données dans la Table 4.1. On retrouve les règles habituelles de la logique intuitionniste du second ordre (règle axiome, règles d'introduction/d'élimination de l'implication et des quantifications universelles), plus la loi de Peirce, qui permet de recouvrer toute l'expressivité de la logique classique [9].

Certaines règles comportent une *condition de bord* (située à droite du trait d'inférence) qui en restreint l'usage. Par exemple, les règles d'intro-

9. Il est plus courant d'introduire le raisonnement classique à l'aide de la loi du tiers-exclu $A \vee \neg A$ ou la règle d'élimination de la double négation $\neg\neg A \Rightarrow A$ (toutes deux équivalentes à la loi de Peirce en logique intuitionniste). Si l'on a choisi la loi de Peirce ici, c'est parce que d'un point de vue calculatoire, elle s'interprète naturellement à l'aide de l'opérateur de contrôle call/cc que nous définirons à la Section 4.3.

$$\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{}{\Gamma_3 \vdash \forall x\,(Q(x) \Rightarrow R(x))}\,\text{(axiome)}}{\Gamma_3 \vdash Q(x) \Rightarrow R(x)}\,(\forall^1\text{-élim}) \quad \dfrac{\dfrac{\dfrac{}{\Gamma_3 \vdash \forall x\,(P(x) \Rightarrow Q(x))}\,\text{(axiome)}}{\Gamma_3 \vdash P(x) \Rightarrow Q(x)}\,(\forall^1\text{-élim}) \quad \dfrac{}{\Gamma_3 \vdash P(x)}\,\text{(axiome)}}{\Gamma_3 \vdash Q(x)}\,(\Rightarrow\text{-élim})}{\Gamma_3 \vdash R(x)}\,(\Rightarrow\text{-élim})}{\Gamma_2 \vdash P(x) \Rightarrow R(x)}\,(\Rightarrow\text{-intro})}{\Gamma_2 \vdash \forall x\,(P(x) \Rightarrow R(x))}\,(\forall^1\text{-intro})}{\Gamma_1 \vdash \forall x\,(Q(x) \Rightarrow R(x)) \Rightarrow \forall x\,(P(x) \Rightarrow R(x))}\,(\Rightarrow\text{-intro})}{\vdash \forall x\,(P(x) \Rightarrow Q(y)) \Rightarrow \forall x\,(Q(x) \Rightarrow R(x)) \Rightarrow \forall x\,(P(x) \Rightarrow R(x))}\,(\Rightarrow\text{-intro})$$

avec $\Gamma_1 \equiv \forall x\,(P(x) \Rightarrow Q(x))$, $\Gamma_2 \equiv \Gamma_1, \forall x\,(Q(x) \Rightarrow R(x))$, $\Gamma_3 \equiv \Gamma_2, P(x)$

TABLE 4.2 – *Exemple de dérivation (syllogisme Barbara)*

duction des quantifications universelles (du premier et du second ordre) imposent que la variable quantifiée n'ait aucune occurrence libre dans le contexte courant ; intuitivement, l'objet que la variable désigne doit être « quelconque », c'est-à-dire : sans hypothèse sur cet objet.

Dérivations Il est commode de voir chaque règle de déduction comme une brique de raisonnement, avec une prise « mâle » [10] pour la conclusion (située au-dessous du trait d'inférence), et autant de prises « femelles » pour les prémisses (situées au-dessus). Ces briques s'assemblent naturellement pour former des *dérivations*, c'est-à-dire des assemblages finis (d'instances) de règles, dans lesquels ne subsiste qu'une seule prise « mâle » libre, située tout en bas de la dérivation, et dont le séquent constitue la *conclusion*.

On trouvera dans la Table 4.2 un exemple de dérivation (purement intuitionniste) du syllogisme Barbara, qui exprime la transitivité de l'inclusion. Cet exemple suffit à rappeler que les dérivations — même de formules très simples — sont des objets en général fort encombrants que l'on construit rarement à la main, mais le plus souvent à l'aide d'assistants à la démonstration, tels que l'assistant Coq par exemple [11].

Expressivité Le système décrit dans la Table 4.1 permet de déduire les règles de raisonnement habituelles associées au connecteurs et aux quanti-

10. Dans cette analogie, le fichage « mâle »/« femelle » est purement conventionnel.

11. La technique de la réalisabilité classique s'étend sans difficulté [86] au *calcul des constructions inductives*, le formalisme de Coq [127].

ficateurs définis (Section 4.2.3), ainsi qu'à l'égalité de Leibniz :

$$\bot \Rightarrow A, \qquad A \Rightarrow B \Rightarrow A \wedge B, \qquad A \wedge B \Rightarrow A, \qquad A \wedge B \Rightarrow B,$$
$$A \Rightarrow A \vee B, \qquad B \Rightarrow A \vee B, \qquad (A \Rightarrow C) \Rightarrow (B \Rightarrow C) \Rightarrow A \vee B \Rightarrow C,$$
$$A(e) \Rightarrow \exists x\, A(x), \qquad \forall x\, (A(x) \Rightarrow C) \Rightarrow \exists x\, A(x) \Rightarrow C,$$
$$A(P) \Rightarrow \exists X\, A(X), \qquad \forall X\, (A(X) \Rightarrow C) \Rightarrow \exists X\, A(X) \Rightarrow C,$$
$$e = e, \qquad e_1 = e_2 \Rightarrow e_2 = e_1, \qquad e_1 = e_2 \Rightarrow e_2 = e_3 \Rightarrow e_1 = e_3, \qquad \text{etc.}$$

La loi de Peirce permet en outre de dériver les règles de raisonnement usuelles de la logique classique : tiers-exclu $A \vee \neg A$, double négation $\neg\neg A \Leftrightarrow A$, contraposition $(A \Rightarrow B) \Leftrightarrow (\neg B \Rightarrow \neg A)$, ainsi que les lois de De Morgan : $\neg(A \wedge B) \Leftrightarrow \neg A \vee \neg B$, $\neg\forall x\, A(x) \Leftrightarrow \exists x\, \neg A(x)$, etc.

Enfin, les règles associées aux quantifications du second ordre permettent de dériver le *schéma de compréhension* (au sens des relations)

$$\exists X\, \forall x_1 \cdots \forall x_k\, (X(x_1, \ldots, x_k) \Leftrightarrow A(x_1, \ldots, x_k))$$

pour chaque formule $A(x_1, \ldots, x_k)$ dépendant de variables $x_1, \ldots, x_k$.

4.2.6 Arithmétique du second ordre (PA2)

L'*arithmétique de Peano du second ordre* (PA2) s'obtient en ajoutant au système décrit précédemment les axiomes suivants :

1. Les axiomes 3 et 4 de Peano, qui expriment que 0 n'est pas un successeur et que la fonction successeur est injective :

$$\forall x\, \neg(s(x) = 0), \qquad \forall x\, \forall y\, (s(x) = s(y) \Rightarrow x = y);$$

2. pour tous les autres symboles de fonction déclarés dans la signature Σ ($+$, $-$, $\times$, $\uparrow$, etc.), les axiomes définitionnels correspondants :

$$\forall x\, (x + 0 = x), \qquad \forall x\, (x \times 0 = 0),$$
$$\forall x\, \forall y\, (x + s(y) = s(x + y)), \quad \forall x\, \forall y\, (x \times s(y) = x \times y + x), \qquad \text{etc.}$$

L'ensemble des axiomes de PA2 est noté $\mathrm{Ax}(\mathrm{PA2})$.

Définition 4.2.1 (Théorème de PA2)**.** — *Une formule close A est un* théorème de l'arithmétique du second ordre *(notation :* $\mathrm{PA2} \vdash A$*) s'il existe une liste finie d'axiomes $\Gamma \subset \mathrm{Ax}(\mathrm{PA2})$ telle que le séquent $\Gamma \vdash A$ soit dérivable.*

On notera cependant que le système d'axiomes de PA2 ne comporte aucun principe d'induction. En logique du second ordre, cette absence n'est pas un problème, car elle signifie simplement que l'ensemble des individus est potentiellement *plus grand* que l'ensemble des entiers naturels. Et comme l'ensemble des entiers naturels est définissable au second ordre par le prédicat de Dedekind (voir Section 4.2.4)

$$\mathbf{N}(x) \;\equiv\; \forall Z\,(Z(0) \Rightarrow \forall y\,(Z(y) \Rightarrow Z(s(y))) \Rightarrow Z(x)),$$

il est toujours possible de remplacer les quantifications du premier ordre par des *quantifications arithmétiques*, c'est-à-dire par des quantifications du premier ordre relativisées au prédicat $\mathbf{N}$ (qui capture les entiers) :

$$(\forall x \in \mathbf{N})A(x) \;\equiv\; \forall x\,(\mathbf{N}(x) \Rightarrow A(x)).$$

Ce mécanisme de relativisation permet ainsi de recouvrer le principe d'induction sous sa forme relativisée

$$\textsc{Ind}^{\mathbf{N}} \;\equiv\; \forall Z\,(Z(0) \Rightarrow (\forall x \in \mathbf{N})(Z(x) \Rightarrow Z(s(x))) \Rightarrow (\forall x \in \mathbf{N})\,Z(x)),$$

formule qui n'est pas un axiome, mais une simple tautologie du système NK2, découlant de la définition même du prédicat $\mathbf{N}$.

En pratique, les quantifications arithmétiques s'utilisent de la même manière que les quantifications non relativisées, dans la mesure où l'ensemble des entiers naturel $\mathbf{N}$ est clos par toutes les opérations qui définissent le langage des termes du premier ordre (Section 4.2.2) :

Proposition 4.2.2 (Totalité des expressions arithmétiques)**.** — *Pour toute expression arithmétique $e(x_1, \ldots, x_k)$ à k variables libres $x_1, \ldots, x_k$, la formule*

$$\mathsf{Total}(e) \;\equiv\; (\forall x_1, \ldots, x_k \in \mathbf{N})\, e(x_1, \ldots, x_k) \in \mathbf{N}$$

est dérivable dans le système NK2 *(sans axiome).*

En particulier, toute expression arithmétique close e est (prouvablement) un entier naturel dans PA2 : $\text{PA2} \vdash e \in \mathbf{N}$.

4.2.7 Le modèle standard de PA2

L'arithmétique du second ordre a un modèle de Tarski très simple, appelé *modèle standard de* PA2 et noté $\mathcal{M}$, dans lequel les individus sont interprétés par les éléments de $\mathbb{N}$ tandis que les relations d'arité k sont interprétées par tous les sous ensembles $R \subseteq \mathbb{N}^k$. Formellement, on appelle une *valuation dans* $\mathcal{M}$ toute fonction ρ qui :

— à chaque variable x du premier ordre associe un entier $\rho(x) \in \mathbb{N}$;
— à chaque variable X du second ordre d'arité k associe un sous-ensemble $\rho(X) \subseteq \mathbb{N}^k$, c'est-à-dire une relation du modèle.

Une *formule à paramètres* dans $\mathcal{M}$ est un couple $A[\rho]$ formé par une formule A et une valuation ρ (dans $\mathcal{M}$). La *relation de satisfaction* dans le modèle $\mathcal{M}$, notée $\mathcal{M} \models A[\rho]$, est alors définie de la manière habituelle :

$$\begin{aligned}
\mathcal{M} \models X(e_1, \dots, e_k)[\rho] &\text{ ssi } (e_1[\rho]^{\mathbb{N}}, \dots, e_k[\rho]^{\mathbb{N}}) \in \rho(X),\\
\mathcal{M} \models (A \Rightarrow B)[\rho] &\text{ ssi } \mathcal{M} \models A[\rho] \text{ implique } \mathcal{M} \models B[\rho],\\
\mathcal{M} \models (\forall x\, A)[\rho] &\text{ ssi } \mathcal{M} \models A[\rho, x \leftarrow n] \text{ pour tout } n \in \mathbb{N},\\
\mathcal{M} \models (\forall X\, A)[\rho] &\text{ ssi } \mathcal{M} \models A[\rho, X \leftarrow R] \text{ pour tout } R \in \mathfrak{P}(\mathbb{N}^k).
\end{aligned}$$

Rappelons que dans la définition ci-dessus :

— $e_i[\rho]^{\mathbb{N}}$ désigne la valeur de l'expression arithmétique e_i dans laquelle chaque variable x est interprétée par l'entier $\rho(x) \in \mathbb{N}$;
— $\rho, x \leftarrow n$ (resp. $\rho, X \leftarrow R$) désigne la valuation obtenue en remplaçant dans ρ la valeur associée à x (resp. à X) par l'entier n (resp. par la relation R), l'ancienne valeur étant écrasée.

Rappelons également que la relation $\mathcal{M} \models A[\rho]$ ne dépend que des valeurs (entiers ou relations) associées aux variables libres de A dans la valuation ρ. Aussi, quand la formule A est close, la relation de satisfaction $\mathcal{M} \models A[\rho]$ ne dépend pas de la valuation ρ, et on la note plus simplement $\mathcal{M} \models A$.

On vérifie sans difficulté que la relation de satisfaction $\mathcal{M} \models A[\rho]$ valide toutes les règles de déduction du système NK2 ainsi que tous les axiomes de l'arithmétique du second ordre (PA2), d'où il ressort que :

Théorème 4.2.3. — *Si* PA2 $\vdash A$ *(A formule close), alors* $\mathcal{M} \models A$.

4.3 Programmes extraits

Le système logique de travail (PA2) étant défini, il s'agit à présent de montrer comment un programme peut être extrait à partir de chaque preuve formalisée dans ce système. Pour cela, nous aurons besoin d'un langage de programmation suffisamment expressif pour traiter les aspects non constructifs de la logique : le λ_c-calcul de Krivine [71].

4.3.1 Termes, piles et processus

Le λ_c-calcul est un langage à pile basé sur le λ-calcul [18, 6], dans lequel on distingue deux catégories syntaxiques : les *termes* (notation : t, u, v, etc.), qui représentent les programmes, et les *piles* (notation : π, π', etc.),

qui représentent les contextes d'évaluation. Formellement, les termes et les piles du λ_c-calcul sont définis à partir d'un ensemble dénombrable de *λ-variables*, notées x, y, z, etc. [12].

Les *termes* du λ_c-calcul sont les termes du λ-calcul enrichis avec des constantes de deux formes : d'une part les deux *instructions* $\mathfrak{cc}$ (« call/cc », pour *call with current continuation*) et stop ; d'autre part les *constantes de continuation* k_π, une pour chaque pile π :

$$\textbf{Termes} \qquad t, u \quad ::= \quad x \quad | \quad \lambda x\,.\,t \quad | \quad tu \quad | \quad \mathfrak{cc} \quad | \quad \mathsf{stop} \quad | \quad \mathsf{k}_\pi.$$

On rappelle que dans la construction $\lambda x\,.\,t$, le symbole λ est un lieur qui lie toutes les occurrences libres de la variable x dans le terme t [13]. L'ensemble des variables libres d'un terme t est noté $FV(t)$.

Enfin, les *piles* du λ_c-calcul sont des listes finies de termes clos (la pile vide étant notée $\diamond$) :

$$\textbf{Piles} \qquad \pi \quad ::= \quad \diamond \quad | \quad t \cdot \pi. \qquad (t \text{ clos})$$

Formellement, les termes et les piles sont donc définis par induction mutuelle à partir des deux grammaires ci-dessus. On notera que contrairement aux termes, les piles sont toujours des objets clos, ce qui permet de représenter chacune d'entre elles par une constante (k_π) dans le monde des termes. L'ensemble des termes clos (resp. des piles) est noté Λ (resp. Π).

4.3.2 La machine abstraite de Krivine (KAM)

Isolément, les termes du λ_c-calcul sont des objets calculatoirement inertes, qui ne peuvent être *évalués* qu'en face d'une pile donnée. Formellement, on appelle un *processus* (ou une *commande*) tout couple $t \star \pi$ formé par un terme clos t et une pile π :

$$\textbf{Processus} \qquad p, q \quad ::= \quad t \star \pi. \qquad (t \in \Lambda,\, \pi \in \Pi)$$

L'ensemble des processus est noté $\Lambda \star \Pi$.

12. Afin d'alléger les notations, on utilise ici les mêmes lettres x, y, z, etc. pour désigner les λ-variables et les variables du premier ordre (Section 4.2.2), bien qu'il s'agisse en réalité de deux espaces de nommage disjoints.

13. Rappelons également que la construction $\lambda x\,.\,t$ (« λ-abstraction ») représente la fonction qui à x associe l'expression t (dépendant de x), tandis que la construction tu (« application ») désigne l'application de la fonction t à l'argument u. Comme en λ-calcul, l'application est notée associativement à gauche, et $t\,u_1 u_2 \cdots u_k$ doit se lire $(\cdots((t\,u_1)u_2)\cdots)u_k$. Les abstractions multiples sont notées $\lambda x_1 x_2 \cdots x_k\,.\,t$ au lieu de $\lambda x_1\,.\,\lambda x_2\,.\,\cdots\lambda x_k\,.\,t$.

Les processus sont évalués au sein de la *machine abstraite de Krivine* (KAM), qui est gouvernée par les quatre règles d'évaluation suivantes :

PUSH	$tu \star \pi$	$\succ$	$t \star u \cdot \pi$
GRAB	$\lambda x\,.\,t \star u \cdot \pi$	$\succ$	$t\{x := u\} \star \pi$
SAVE	$\mathfrak{cc} \star u \cdot \pi$	$\succ$	$u \star \mathsf{k}_\pi \cdot \pi$
RESTORE	$\mathsf{k}_\pi \star u \cdot \pi'$	$\succ$	$u \star \pi$.

Formellement, la relation d'évaluation $p \succ p'$ («p s'évalue sur p'») est définie comme le préordre — c'est-à-dire comme la relation réflexive et transitive — engendré par les quatre règles ci-dessus. (Comme ces quatre règles ont des membres gauches deux à deux disjoints, la relation d'évaluation qu'elles engendrent est parfaitement déterministe.)

Anatomie des règles d'évaluation Les deux premières règles implémentent la partie purement « intuitionniste » du calcul :

— PUSH : L'application tu empile l'argument u au sommet de la pile, avant de passer la main au terme t (la « fonction »).

— GRAB : L'abstraction $\lambda x\,.\,t$ dépile l'argument u au sommet de la pile, le substitue à la variable x dans le corps t de la fonction, avant de passer la main au terme substitué $t\{x := u\}$.

On notera que l'argument u n'est pas évalué au cours d'un GRAB ou d'un PUSH. La KAM a en effet une stratégie d'évaluation en *appel par nom*, suivant laquelle les arguments ne sont évalués que lorsqu'ils arrivent en position de tête — c'est-à-dire à gauche du symbole $\star$[14].

Les deux règles suivantes implémentent un mécanisme très simple de sauvegarde et de restauration de la pile courante (mécanisme de *backtrack*), qui constitue la partie « classique » du calcul :

— SAVE : L'instruction $\mathfrak{cc}$ (« call/cc ») dépile le terme u au sommet de la pile (qu'on peut voir comme une *adresse de retour*), effectue une copie de la pile courante π sous la forme d'une constante de continuation k_π, et place cette constante de continuation au sommet de la pile avant de passer la main au terme u.

— RESTORE : La constante de continuation k_π (issue d'un SAVE antérieur) dépile le terme u (l'adresse de retour) au sommet de la pile, remplace la pile courante (π') par la pile π sauvegardée (*backtrack*), avant de passer la main au terme u.

En pratique, l'instruction call/cc s'utilise le plus souvent dans la tournure idiomatique $\mathfrak{cc}\,(\lambda k\,.\,t)$, qui signifie : « sauver la pile courante dans la

14. Du point de vue du λ-calcul, ces deux règles implémentent une stratégie d'évaluation connue sous le nom de *β-réduction de tête faible*.

variable k avant d'évaluer t » :

$$\begin{array}{rcll} \mathrm{cc}\,(\lambda k\,.\,t)\star\pi & \succ & \mathrm{cc}\star(\lambda k\,.\,t)\cdot\pi & (\textsc{Push}) \\ & \succ & \lambda k\,.\,t\star \mathsf{k}_\pi\cdot\pi & (\textsc{Save}) \\ & \succ & t\{k:=\mathsf{k}_\pi\}\star\pi. & (\textsc{Grab}) \end{array}$$

Une fois la constante k_π capturée dans la variable k, celle-ci peut être mise en réserve et/ou passée à d'autres fonctions (et cela récursivement), jusqu'au moment où un terme quelconque — évalué dans un tout autre contexte π' — réinstalle la pile sauvegardée, en appliquant la variable k au terme u désiré (qui peut lui-même dépendre de la variable k) :

$$\begin{array}{rcll} t\{k:=\mathsf{k}_\pi\}\star\pi\ \succ\ \cdots & \succ & (k\,u)\{k:=\mathsf{k}_\pi\}\star\pi' & \\ & \succ & \mathsf{k}_\pi\star u\{k:=\mathsf{k}_\pi\}\cdot\pi' & (\textsc{Push}) \\ & \succ & u\{k:=\mathsf{k}_\pi\}\star\pi. & (\textsc{Restore}) \end{array}$$

L'évaluation du terme u se poursuit alors dans le contexte π restauré.

Enfin, l'instruction stop n'a pas de règle d'évaluation, sa seule fonction étant précisément de stopper l'exécution de la KAM.

4.3.3 Un système de types pour la logique du second ordre

Afin d'expliciter le lien entre la logique et le calcul, nous allons maintenant transformer le système de *déduction* NK2 (Section 4.2.5 et Table 4.1) en un système de *types*, noté λNK2, dont les types seront précisément les formules de l'arithmétique du second ordre. Formellement, ce système de types s'articule autour d'un *jugement de typage* de la forme

$$\Gamma \vdash t : A$$

(« dans le contexte Γ, le terme t a le type A »), où Γ est un contexte de typage, t un terme du λ_c-calcul, et A une formule quelconque.

Un *contexte de typage* est une liste finie de déclarations de la forme

$$\Gamma \ \equiv\ x_1 : A_1, \ldots, x_n : A_n$$

où $x_1, \ldots, x_n$ sont des λ-variables deux à deux distinctes, et où $A_1, \ldots, A_n$ sont des formules. Pour tout contexte de typage $\Gamma \equiv x_1 : A_1, \ldots, x_n : A_n$, on note $\mathrm{dom}(\Gamma) = \{x_1, \ldots, x_n\}$ son *domaine* (il s'agit d'un ensemble fini de λ-variables), et suivant l'usage, on écrit $FV(\Gamma) = FV(A_1) \cup \cdots \cup FV(A_n)$ l'ensemble de ses *variables libres* (il s'agit d'un ensemble fini de variables logiques, du premier et du second ordre).

Les règles de typage du système λNK2 sont données dans la Table 4.3. Comme pour les règles de déduction du système NK2, les règles de typage du système λNK2 s'assemblent pour former des *dérivations de typage*, dont la conclusion est un *jugement dérivable*.

(Axiome)	$\dfrac{}{\Gamma \vdash x : A}$ si $(x:A)\in\Gamma$
($\Rightarrow$-intro/elim)	$\dfrac{\Gamma, x : A \vdash t : B}{\Gamma \vdash \lambda x\,.\,t : A \Rightarrow B}$ $\qquad$ $\dfrac{\Gamma \vdash t : A \Rightarrow B \qquad \Gamma \vdash u : A}{\Gamma \vdash t\,u : B}$
($\forall^1$-intro/élim)	$\dfrac{\Gamma \vdash t : A}{\Gamma \vdash t : \forall x\, A}$ si $x \notin FV(\Gamma)$ $\qquad$ $\dfrac{\Gamma \vdash t : \forall x\, A}{\Gamma \vdash t : A\{x := e\}}$
($\forall^2$-intro/élim)	$\dfrac{\Gamma \vdash t : A}{\Gamma \vdash t : \forall X A}$ si $X \notin FV(\Gamma)$ $\qquad$ $\dfrac{\Gamma \vdash t : \forall X A}{\Gamma \vdash t : A\{X := P\}}$
(Loi de Peirce)	$\dfrac{}{\Gamma \vdash \mathfrak{c} : ((A \Rightarrow B) \Rightarrow A) \Rightarrow A}$

TABLE 4.3 – *Règles de typage du système* λNK2

Typage et déduction Du point de vue de la logique, un jugement de typage $x_1 : A_1, \ldots, x_n : A_n \vdash t : A$ n'est rien d'autre qu'un séquent (Section 4.2.5) décoré avec de l'information calculatoire, dans lequel :

— chaque hypothèse A_i ($i = 1, \ldots, n$) est annotée avec une λ-variable x_i, qui représente une « preuve générique » de la formule A_i ;

— la conclusion A est annotée avec un λ_c-terme t (dépendant des λ-variables $x_1, \ldots, x_n$), appelé *terme de preuve* de la formule A, lequel représente le contenu calculatoire d'une certaine preuve de A.

Pour formaliser le lien entre les systèmes λNK2 et NK2, on introduit une fonction d'*effacement* du premier vers le second, chargée d'effacer l'information calculatoire pour ne conserver que l'information logique.

Ainsi à chaque contexte de typage $\Gamma \equiv x_1 : A_1, \ldots, x_n : A_n$ on associe le contexte logique $\Gamma^* \equiv A_1, \ldots, A_n$, et à chaque jugement de typage $\Gamma \vdash t : A$ le séquent logique $(\Gamma \vdash t : A)^* \equiv \Gamma^* \vdash A$ qui lui correspond. Et comme à travers la fonction d'effacement, les règles de typage de la Table 4.3 correspondent exactement aux règles de déduction de la Table 4.1, la fonction d'effacement s'étend naturellement aux dérivations de typage, qu'elle transforme en dérivations logiques.

On vérifie alors sans difficulté qu'à travers la fonction d'effacement, les systèmes NK2 et λNK2 sont équivalents, en ce sens que :

Proposition 4.3.1 (Équivalence des systèmes NK2 et λNK2)**.**

1. *Si d est une dérivation de typage d'un jugement $\Gamma \vdash t : A$, alors d^* est une dérivation logique du séquent $\Gamma^* \vdash A$.*

2. *Toute dérivation logique d d'un séquent* $\Gamma \vdash A$ *provient d'une dérivation de typage* d_0 *d'un jugement de typage de la forme* $\Gamma_0 \vdash t : A$, *avec* $d_0^* = d$ *et* $\Gamma_0^* = \Gamma$. *De plus, la dérivation de typage* d_0 *et le terme de preuve t sont uniques, aux noms de variables près* [15]. *On dit alors que le terme de preuve t est le* programme extrait *à partir de la preuve d.*

En pratique, le programme extrait t est construit par récursion sur la structure de la dérivation logique d, en utilisant à chaque étape de déduction la λ-construction indiquée par la règle de typage correspondante : une variable pour la règle axiome, une λ-abstraction pour la règle $\Rightarrow$-intro, une application pour la règle $\Rightarrow$-élim, etc.

Exemple 4.3.2. — *Si dans la dérivation de la Table 4.2 (syllogisme Barbara) on associe aux trois hypothèses* $\forall x\,(P(x) \Rightarrow Q(x))$, $\forall x\,(Q(x) \Rightarrow R(x))$ *et* $P(x)$ *les trois* λ*-variables* f, g *et* z, *alors le programme extrait à partir de cette dérivation est le* λ_c*-terme* $\lambda f g z\,.\,g\,(f\,z)$ *(opérateur de composition de fonctions).*

Typage et programmation Du point de vue de la programmation, le système de types de la Table. 4.3 traite l'implication $A \Rightarrow B$ comme un type de fonctions. En particulier, la règle $\Rightarrow$-élim (le *modus ponens* de la logique) correspond exactement à la règle de typage habituelle de l'application d'une fonction (de type $A \Rightarrow B$) à son argument (de type A).

On notera que contrairement aux autres règles de la Table. 4.3, les règles de typage des quantifications universelles du premier et du second ordre ne sont associées à aucune construction du λ_c-calcul. En réalisabilité classique en effet, la quantification universelle est traitée comme un *type intersection* (infinitaire) : un terme t a le type $\forall x\,A(x)$ (resp. $\forall X A(X)$) si et seulement si le même terme t a le type $A(x)$ (resp. $A(X)$) pour tout individu x (resp. pour toute relation X). Les règles de typage correspondantes sont donc les mêmes que celles qui traitent le *polymorphisme* [16] dans les langages fonctionnels (Caml, SML, Haskell, etc.)

Enfin, l'instruction $\mathfrak{c}$ (« call/cc ») a pour type la loi de Peirce, ce sur quoi nous reviendrons plus en détail à la Section 4.4.

4.3.4 Exemples de termes de preuve

Le système de types λNK2 permet d'introduire un certain nombre de constructions génériques qui seront utiles dans la suite :

15. En effet, la dérivation de typage d_0 comme le terme de preuve t dépendent des λ-variables utilisées pour annoter toutes les hypothèses figurant dans la dérivation d.

16. La quantification du second ordre $\forall X\,A(X)$ sur les relations d'arité 0 (les propositions) correspond exactement à la quantification de type des langages fonctionnels.

— Le constructeur de paire et les deux projections associées à la conjonction $A \wedge B$ (type *produit cartésien*) :

$$\begin{array}{rcl} \langle t, u\rangle \equiv \lambda f\,.\,f\,x\,y & : & A \wedge B \quad (\text{si } t : A, u : B);\\ \mathsf{pair} \equiv \lambda xy\,.\,\langle x, y\rangle & : & \forall X\,\forall Y\,(X \Rightarrow Y \Rightarrow X \wedge Y);\\ \mathsf{fst} \equiv \lambda z\,.\,z\,(\lambda xy\,.\,x) & : & \forall X\,\forall Y\,(X \wedge Y \Rightarrow X);\\ \mathsf{snd} \equiv \lambda z\,.\,z\,(\lambda xy\,.\,y) & : & \forall X\,\forall Y\,(X \wedge Y \Rightarrow Y). \end{array}$$

— Les injections associées à la disjonction $A \vee B$ (type *somme directe*) :

$$\begin{array}{rcl} \mathsf{left} \equiv \lambda xfg\,.\,f\,x & : & \forall X\,\forall Y\,(X \Rightarrow X \vee Y);\\ \mathsf{right} \equiv \lambda yfg\,.\,g\,y & : & \forall X\,\forall Y\,(Y \Rightarrow X \vee Y). \end{array}$$

Interprétation calculatoire du tiers-exclu Un exemple typique d'utilisation de l'instruction ∝ pour interpréter calculatoirement un raisonnement non constructif est donné par la preuve suivante du tiers-exclu

$$\mathsf{EM} \;\equiv\; \propto(\lambda k\,.\,\mathsf{right}\,(\lambda x\,.\,k\,(\mathsf{left}\,x))) \;:\; \forall X\,(X \vee \neg X)$$

qui implémente ici la stratégie du « prêcher le faux pour savoir le vrai ».

En effet, après avoir capturé la continuation courante (variable k), le terme EM « joue » dans un premier temps le membre droit de la disjonction $X \vee \neg X$, en produisant une preuve factice $\lambda x\,.\,k\,(\mathsf{left}\,x)$ de la formule $\neg X$ ($\equiv X \Rightarrow \bot$) dont le seul but est de défier l'opposant à produire une preuve de X (coup de *bluff*). Si au cours de l'évaluation, l'opposant est effectivement en mesure de passer une preuve de X (par la variable x) à la preuve factice de $X \Rightarrow \bot$, alors celle-ci effectue un *backtrack* à l'aide de la continuation cachée dans la variable k pour revenir à la situation de départ. Ce qui permet au joueur (EM) de donner une nouvelle preuve de la disjonction $X \vee \neg X$, mais en jouant cette fois-ci le membre gauche…avec la preuve $x : X$ fournie par l'opposant [17].

Les entiers du calcul En λ-calcul [6], on représente traditionnellement l'entier 0 et la fonction successeur (sur les entiers naturels) par les λ-termes clos $\overline{0} \equiv \lambda zf\,.\,z$ et $\overline{s} \equiv \lambda nzf\,.\,f\,(n\,z\,f)$. Dans le système λNK2, ces deux

17. Ce que cette situation a de piquant, c'est que la preuve $x : X$ fournie par l'opposant peut elle-même être un coup de bluff destiné à piéger le terme EM. Ainsi, la logique classique apparaît comme un jeu de bluff/contre-bluff permanent, dont la terminaison sera assurée par le modèle de réalisabilité de la Section 4.4 et les résultats de la Section 4.5.3.

$\lambda_{(c)}$-termes sont également des termes de preuve des formules exprimant que l'ensemble $\mathbf{N}$ (Section 4.2.4) contient 0 et est clos par successeur :

$$\begin{array}{lll} \overline{0} \equiv \lambda z f\,.\,z & : & 0 \in \mathbf{N}, \\ \overline{s} \equiv \lambda n z f\,.\,f\,(n\,z\,f) & : & (\forall x \in \mathbf{N})\, s(x) \in \mathbf{N}. \end{array}$$

En λ_c-calcul, on représente ainsi chaque entier naturel $n \in \mathbb{N}$ par l'*entier de Krivine*

$$\overline{n} :\equiv \overline{s}^n\,\overline{0} \equiv \underbrace{\overline{s}\,(\cdots(\overline{s}}_{n}\,\overline{0})\cdots), \qquad n \in \mathbf{N},$$

qui constitue un terme de preuve de la formule $n \in \mathbf{N}$ ($\equiv s^n(0) \in \mathbf{N}$ [18]).

D'un point de vue calculatoire, l'entier de Krivine $\overline{n} \equiv \overline{s}^n\,\overline{0}$ se comporte comme un *itérateur*, qui, lorsqu'on lui passe deux arguments quelconques $u_0, u_1 \in \Lambda$, applique n fois de suite u_1 sur u_0 :

$$\begin{array}{rcl} \overline{0} \star u_0 \cdot u_1 \cdot \pi & \succ & u_0 \star \pi, \\ \overline{n+1} \star u_0 \cdot u_1 \cdot \pi & \succ & u_1 \star (\overline{n}\,u_0\,u_1) \cdot \pi. \end{array} \qquad (\pi \in \Pi)$$

4.4 Le modèle de réalisabilité classique

Les règles de typage de la Table 4.3 permettent d'associer à chaque dérivation logique du système NK2 un certain *terme de preuve* (ou *programme extrait*), mais elles ne permettent pas à ce stade de prédire ce que ce terme de preuve calcule. Afin de déterminer le comportement calculatoire des termes extraits, nous allons maintenant construire un modèle d'un type particulier, qu'on appelle un *modèle de réalisabilité*.

À la différence d'un modèle de Tarski, où chaque formule A est interprétée par un booléen $|A| \in \{0,1\}$, un modèle de réalisabilité interprète chaque formule A par un ensemble de termes clos $|A| \subseteq \Lambda$, appelés *réalisateurs* de A, lesquels partagent un même comportement calculatoire défini par induction sur la structure de A. Le lien entre typage et calcul sera alors donné par le *théorème d'adéquation*, qui exprime que tout terme de preuve t de la formule A satisfait la spécification attendue : $t \in |A|$.

4.4.1 Architecture du modèle

Pôles, valeurs de vérité, valeurs de fausseté La réalisabilité classique [71] se distingue de la réalisabilité intuitionniste [68, 70] par le

18. Comme d'habitude en arithmétique, on identifie chaque entier naturel $n \in \mathbb{N}$ (entier « externe » au langage) avec l'entier de Peano correspondant, c'est-à-dire avec l'expression arithmétique close $s^n(0)$ (entier « interne » au langage). On veillera à ne pas confondre cet entier « logique » avec l'entier « calculatoire » $\overline{n} \equiv \overline{s}^n\,\overline{0}$ (qui est un λ_c-terme clos).

fait qu'elle décrit le comportement calculatoire associé à chaque formule A à travers l'interaction entre les termes chargés de *défendre* la formule (les *réalisateurs* de A) et les piles chargées de l'*attaquer* (les *contre-réalisateurs* de A).

Formellement, la construction du modèle de réalisabilité est paramétrée par un ensemble de processus $\bot\!\!\!\bot \subseteq \Lambda \star \Pi$, le *pôle* du modèle, qui représente intuitivement une certaine règle du jeu, c'est-à-dire une condition de victoire d'un terme (défendant une certaine formule) contre une pile (s'opposant à la même formule). On suppose que le pôle $\bot\!\!\!\bot \subseteq \Lambda \star \Pi$ est *clos par anti-évaluation*, en ce sens que pour tous processus $p, p' \in \Lambda \star \Pi$, les conditions $p \succ p'$ et $p' \in \bot\!\!\!\bot$ entraînent que $p \in \bot\!\!\!\bot$. Le modèle de réalisabilité associé au pôle $\bot\!\!\!\bot$ est noté $\mathcal{M}_{\bot\!\!\!\bot}$.

Dans le modèle $\mathcal{M}_{\bot\!\!\!\bot}$, la sémantique d'une formule est définie à partir d'une *valeur de fausseté*, c'est-à-dire un ensemble de piles $S \subseteq \Pi$ capturant toutes les attaques possibles de la formule considérée. Chaque valeur de fausseté $S \subseteq \Pi$ induit une *valeur de vérité* $S^{\bot\!\!\!\bot} \subseteq \Lambda$, dont les éléments sont par définition les λ_c-termes capables de « déjouer » (au sens du pôle $\bot\!\!\!\bot$) toutes les attaques issues de l'ensemble S :

$$S^{\bot\!\!\!\bot} := \{t \in \Lambda \mid \forall \pi \in S, t \star \pi \in \bot\!\!\!\bot\}. \qquad (\subseteq \Lambda)$$

On notera que plus la valeur de fausseté $S \subseteq \Pi$ est grande, plus la valeur de vérité $S^{\bot\!\!\!\bot} \subseteq \Lambda$ qui lui est associée est petite (et vice-versa).

Valuations et formules avec paramètres Dans le modèle $\mathcal{M}_{\bot\!\!\!\bot}$, les individus sont interprétés par des entiers naturels (exactement de la même manière que dans le modèle standard $\mathcal{M}$ décrit à la Section 4.2.7), tandis que les relations d'arité k sont interprétées par des *fonctions de fausseté* $F : \mathbb{N}^k \to \mathfrak{P}(\Pi)$. Formellement, on appelle une *valuation dans* $\mathcal{M}_{\bot\!\!\!\bot}$ toute fonction ρ (définie sur l'ensemble des variables logiques) qui :

— à chaque variable x du premier ordre associe un entier $\rho(x) \in \mathbb{N}$;
— à chaque variable X du second ordre d'arité k associe une fonction de fausseté $\rho(X) : \mathbb{N}^k \to \mathfrak{P}(\Pi)$ de la même arité.

Enfin, on appelle une *formule avec paramètres (dans* $\mathcal{M}_{\bot\!\!\!\bot}$*)* tout couple $A[\rho]$ formé par une formule A et une valuation ρ (dans $\mathcal{M}_{\bot\!\!\!\bot}$).

Notation 4.4.1. — *Quand on manipule une formule avec paramètres $A[\rho]$ dans le modèle de réalisabilité $\mathcal{M}_{\bot\!\!\!\bot}$, il est commode d'intégrer la valuation ρ dans l'écriture de la formule A, en remplaçant dans celle-ci :*

— *chaque occurrence libre d'une variable x du premier ordre par l'entier $n = \rho(x)$ associé à la variable x dans la valuation ρ, et*

— *chaque occurrence libre d'une variable X du second ordre par un symbole de prédicat fictif $\dot{F}$ (de même arité) chargé de représenter la fonction de fausseté $F = \rho(X)$ associée à la variable X dans la valuation ρ.*

Ainsi, en supposant que ρ est une valuation associant l'entier $\rho(y) = 3$ à la variable y et la fonction de fausseté $\rho(Z) = F : \mathbb{N} \to \mathfrak{P}(\Pi)$ à la variable Z (d'arité 1), on écrira plus simplement

$$\forall x\,(\dot{F}(x) \Rightarrow \dot{F}(x+3)) \quad \textit{pour} \quad (\forall x\,(Z(x) \Rightarrow Z(x+y)))[\rho].$$

4.4.2 La fonction d'interprétation

Définition 4.4.2 (Interprétation des formules)**.** — *À chaque formule avec paramètres $A[\rho]$, le modèle de réalisabilité $\mathcal{M}_{\perp\!\!\!\perp}$ associe deux ensembles :*

— *une* valeur de fausseté, *notée $\|A[\rho]\|$ $(\subseteq \Pi)$; et*

— *une* valeur de vérité, *notée $|A[\rho]|$ $(\subseteq \Lambda)$.*

Ces deux ensembles sont définis par induction mutuelle sur la structure de la formule A à l'aide des équations suivantes :

$$\begin{aligned}
\|X(e_1,\dots,e_k)[\rho]\| &:= \rho(X)(e_1[\rho]^{\mathbb{N}},\dots,e_k[\rho]^{\mathbb{N}}),\\
\|(A \Rightarrow B)[\rho]\| &:= |A[\rho]| \cdot \|B[\rho]\| = \{t \cdot \pi \mid t \in |A[\rho]|, \pi \in \|B[\rho]\|\},\\
\|(\forall x\, A)[\rho]\| &:= \bigcup_{n\in\mathbb{N}} \|A[\rho, x \leftarrow n]\|,\\
\|(\forall X\, A)[\rho]\| &:= \bigcup_{F:\mathbb{N}^k\to\mathfrak{P}(\Pi)} \|A[\rho, X \leftarrow F]\|,\\
|A[\rho]| &:= \|A[\rho]\|^{\perp\!\!\!\perp} = \{t \in \Lambda \mid \forall \pi{\in}\|A[\rho]\|, t \star \pi \in \perp\!\!\!\perp\}.
\end{aligned}$$

Dans la mesure où la valeur de vérité $|A[\rho]|$ et la valeur de fausseté $\|A[\rho]\|$ dépendent fortement du pôle $\perp\!\!\!\perp$ considéré, on utilisera parfois les notations $|A[\rho]|_{\perp\!\!\!\perp}$ et $\|A[\rho]\|_{\perp\!\!\!\perp}$ pour rappeler cette dépendance.

Définition 4.4.3 (Réalisabilité)**.** — *Étant donnés un pôle $\perp\!\!\!\perp$, une formule à paramètres $A[\rho]$ et un λ_c-terme clos t, on dit que t* réalise *$A[\rho]$ et on écrit $t \Vdash A[\rho]$ lorsque $t \in |A[\rho]|_{\perp\!\!\!\perp}$, en gardant à l'esprit que cette écriture dépend du pôle $\perp\!\!\!\perp$. Lorsque $t \in |A[\rho]|_{\perp\!\!\!\perp}$ pour tous les pôles $\perp\!\!\!\perp$, on dit que t* réalise universellement *$A[\rho]$, ce que l'on note $t \Vdash\!\!\!\Vdash A[\rho]$.*

Dans la Section 4.4.3, nous verrons que tous les termes de preuve d'une formule close sont des réalisateurs universels de cette formule, c'est-à-dire, intuitivement, des réalisateurs indépendants de la « règle du jeu ».

Anatomie du modèle En réalisabilité classique, la sémantique d'une formule A avec paramètres est d'abord donnée par sa valeur de fausseté $\|A\|$. Pour comprendre les différentes clauses de la Définition 4.4.2, il convient donc se placer du point de vue de l'opposant. Ainsi :

— L'implication $A \Rightarrow B$ est une formule qu'on attaque avec une pile de la forme $t \cdot \pi$, où t est une défense de la formule A et π une attaque de la formule B. (Cette définition est donc réminiscente de l'équivalence $\neg(A \Rightarrow B) \Leftrightarrow A \wedge \neg B$.) Du point de vue de la programmation, il est clair que les piles de la forme $t \cdot \pi$ avec $t \in |A|$ (tête de type A) et $\pi \in \|B\|$ (queue s'opposant à B) sont les piles adéquates pour évaluer une fonction de type $A \Rightarrow B$.

— La quantification universelle $\forall x\, A(x)$ (resp. $\forall X A(X)$) est une formule qu'on attaque à l'aide d'une pile π attaquant au moins l'une des instances $A(n)$ (resp. $A(\dot{F})$) de la formule quantifiée. Si du point de vue de l'opposant, la sémantique de la quantification universelle correspond naturellement à une union, du point de vue du défenseur, elle correspondra ainsi à l'intersection voulue, puisque :

$$|\forall x\, A(x)| = \Big(\bigcup_n \|A(n)\|\Big)^{\perp\!\!\!\perp} = \bigcap_n \|A(n)\|^{\perp\!\!\!\perp} = \bigcap_n |A(n)|,$$
$$|\forall X A(X)| = \Big(\bigcup_F \|A(\dot{F})\|\Big)^{\perp\!\!\!\perp} = \bigcap_F \|A(\dot{F})\|^{\perp\!\!\!\perp} = \bigcap_F |A(\dot{F})|.$$

Cas particulier du pôle vide Dans le cas particulier où le pôle est vide ($\perp\!\!\!\perp = \varnothing$), il est facile de voir que le modèle de réalisabilité associé se comporte exactement comme le modèle standard $\mathcal{M}$ défini à la Section 4.2.7. En effet, on remarque tout d'abord que lorsque $\perp\!\!\!\perp = \varnothing$, la valeur de vérité $S^{\perp\!\!\!\perp} \subseteq \Lambda$ associée à une valeur de fausseté $S \subseteq \Pi$ quelconque ne peut prendre que deux valeurs possibles : $S^{\perp\!\!\!\perp} = \Lambda$ si $S = \varnothing$, et $S^{\perp\!\!\!\perp} = \varnothing$ sinon. De cette observation, on déduit par une récurrence immédiate que :

Proposition 4.4.4. — *Si $\perp\!\!\!\perp = \varnothing$, alors pour toute formule close A, on a*

$$|A| = \begin{cases} \Lambda & \text{si } \mathcal{M} \models A, \\ \varnothing & \text{si } \mathcal{M} \not\models A. \end{cases}$$

Cette observation implique en particulier que toute formule qui admet au moins un réalisateur universel est vraie dans le modèle standard :

$$t \Vdash A \quad \text{implique} \quad \mathcal{M} \models A.$$

Cas d'un pôle non vide Dans le cas où $\perp\!\!\!\perp \neq \varnothing$, il est facile de voir que toute valeur de vérité $S^{\perp\!\!\!\perp}$ est habitée, par exemple par n'importe quel terme de la forme $\mathsf{k}_\pi t$, où $t \star \pi \in \perp\!\!\!\perp$. Cette remarque suffit à montrer que dans le cas où $\perp\!\!\!\perp \neq \varnothing$, un réalisateur $t \Vdash A$ ne peut en aucun cas être considéré comme un « certificat » attestant de la vérité de la formule A [19]. C'est là une différence fondamentale avec la réalisabilité intuitionniste, qui a des conséquences profondes en ce qui concerne le design des procédures d'extraction de témoin en logique classique (Section 4.5.3).

4.4.3 Le théorème d'adéquation

Il est temps à présent de relier la notion (sémantique) de réalisabilité avec la notion (syntaxique) de typage introduite à la Section 4.3.3.

Définition 4.4.5 (Adéquation)**.** — *Soit $\perp\!\!\!\perp \subseteq \Lambda \times \Pi$ un pôle fixé.*

1. *Un jugement de typage $x_1 : A_1, \ldots, x_n : A_n \vdash t : A$ est* adéquat *(vis-à-vis du pôle $\perp\!\!\!\perp$) si pour toute valuation ρ et pour tous réalisateurs $u_1 \Vdash A_1[\rho], \ldots, u_n \Vdash A_1[\rho]$, on a $t\{x_1 := u_1, \ldots, x_n := u_n\} \Vdash A[\rho]$.*
2. *Une règle de typage est* adéquate *(vis-à-vis du pôle $\perp\!\!\!\perp$) si l'adéquation de toutes ses prémisses implique l'adéquation de sa conclusion.*

Ainsi, un jugement de typage adéquat est un jugement de typage dans lequel on peut substituer toutes les λ-variables libres du terme de preuve par des réalisateurs quelconques des hypothèses correspondantes pour obtenir un réalisateur de la conclusion. (Il s'agit donc d'un « patron » de construction de réalisateurs de la conclusion.) Et comme par définition, les règles de typage adéquates sont celles qui transfèrent cette propriété des prémisses à la conclusion, il est clair que toute dérivation de typage construite en n'utilisant que des règles de typage adéquates ne peut qu'aboutir à un jugement de typage adéquat. On démontre alors le :

Théorème 4.4.6 (Adéquation)**.** — *Toutes les règles de typage du système λ*NK2 *(Table 4.3) sont adéquates vis-à-vis de tout pôle $\perp\!\!\!\perp$, de même que tous les jugements de typage dérivables dans ce système.*

19. Nous avons déjà vu (Section 4.3.4) que les réalisateurs classiques sont susceptibles d'embarquer des constantes de continuation, lesquelles correspondant à des « coups de bluff » de la preuve. De fait, on peut voir les constantes de continuation k_π comme des *parapreuves*, c'est-à-dire comme des preuves entraînant localement l'incohérence du système (incohérence locale absolument cruciale pour extraire un témoin en logique classique). On notera cependant que tous les réalisateurs universels issus des preuves de PA2 (voir Théorème 4.4.11 p. 105) sont dépourvus de toute constante de continuation.

La démonstration (voir par exemple [71, 87, 40]) ne présente aucune difficulté et est laissée en exercice au lecteur. Elle consiste essentiellement à vérifier que chacune des règles de typage de la Table 4.3 préserve la propriété d'adéquation, en évaluant la construction correspondante dans la KAM et en utilisant le fait que le pôle $\perp\!\!\!\perp$ est clos par anti-évaluation. On notera que le traitement de la règle de typage de l'instruction $\mathfrak{c}$ associée à la loi de Peirce $((A \Rightarrow B) \Rightarrow A) \Rightarrow A$ repose sur le lemme suivant :

Lemme 4.4.7. — *Étant données deux formules A et B avec paramètres, pour toute pile $\pi \in \|A\|$, on a $\mathsf{k}_\pi \in |A \Rightarrow B|$.*

(Là encore, la démonstration est laissée en exercice au lecteur.)

Le théorème d'adéquation implique en particulier que si t est un terme de preuve d'une formule close A (sans hypothèse) dans le système λNK2, alors ce terme constitue un réalisateur universel de A : $t \Vdash A$.

4.4.4 Réalisation des théorèmes de PA2

Le théorème d'adéquation ne concerne pour le moment que les dérivations purement logiques (système NK2). Afin d'étendre ce résultat à toutes les preuves dans PA2 (Section 4.2.6), il nous reste à vérifier que tous les axiomes de PA2 ont eux-aussi des réalisateurs universels.

Pour cela, il est commode d'étendre le langage des formules avec une constante propositionnelle $\top$ (formule « triviale »)

Formules $\qquad A, B \quad ::= \quad \cdots \quad | \quad \top$

dont la sémantique est donnée dans n'importe quel pôle $\perp\!\!\!\perp$ par $\|\top\| = \varnothing$ (aucun opposant) et $|\top| = \varnothing^{\perp\!\!\!\perp} = \Lambda$ (tous les termes sont des réalisateurs). Enfin, on note $\mathbf{1} \equiv \forall Z\,(Z \Rightarrow Z)$ le « type » de l'identité $\mathbf{I} \equiv \lambda z\,.\,z$ [20].

Sémantique de l'égalité de Leibniz Rappelons que dans le langage de PA2, l'égalité de Leibniz est définie par $e_1 = e_2 \equiv \forall Z\,(Z(e_1) \Rightarrow Z(e_2))$. La sémantique de cette formule est donnée par le lemme suivant :

Lemme 4.4.8 (Sémantique de l'égalité de Leibniz)**.** — *Soient e_1 et e_2 deux expressions arithmétiques. Pour tout pôle $\perp\!\!\!\perp$ et pour toute valuation ρ, on a :*

$$\|(e_1 = e_2)[\rho]\| = \begin{cases} \{t \cdot \pi \mid t \star \pi \in \perp\!\!\!\perp\} = \|\mathbf{1}\| & \text{si } e_1[\rho]^{\mathbb{N}} = e_2[\rho]^{\mathbb{N}}, \\ \Lambda \cdot \Pi = \|\top \Rightarrow \bot\| & \text{si } e_1[\rho]^{\mathbb{N}} \neq e_2[\rho]^{\mathbb{N}}. \end{cases}$$

(La démonstration résulte d'un simple calcul de dénotations.)

20. On peut même démontrer [40] que les réalisateurs universels de la formule $\mathbf{1} \equiv \forall Z\,(Z \Rightarrow Z)$ sont précisément les λ_c-termes qui se comportent comme la fonction identité.

Intuitivement, le lemme ci-dessus exprime que lorsque l'égalité $e_1 = e_2$ est vraie dans le modèle standard, alors tout réalisateur de cette formule se comporte comme la fonction identité, tandis que lorsqu'elle est fausse, il se comporte comme une fonction de type $\top \Rightarrow \bot$. (Nous verrons à la Section 4.5.3 que ce dernier cas correspond à une situation de *backtrack*.)

De la caractérisation précédente, on déduit immédiatement que toutes les formules Π^0_1 (égalités universellement quantifiées) qui sont vraies dans le modèle standard sont uniformément réalisées par le terme $\mathbf{I} \equiv \lambda z\,.\,z$:

Proposition 4.4.9 (Réalisation des formules Π^0_1)**.** — *Soient* $e_1(x_1, \ldots, x_k)$ *et* $e_2(x_1, \ldots, x_k)$ *deux expressions arithmétiques telles que*

$$\mathcal{M} \models \forall x_1 \cdots \forall x_k\, (e_1(x_1, \ldots, x_k) = e_2(x_1, \ldots, x_k)).$$

Alors, $\quad \lambda z\,.\,z \Vdash \forall x_1 \cdots \forall x_k\, (e_1(x_1, \ldots, x_k) = e_2(x_1, \ldots, x_k)).$

Réalisation des axiomes de PA2 Comme les axiomes définissant les symboles de fonction associés aux fonctions récursives primitives ($+$, $-$, $\times$, etc.) sont des formules Π^0_1 qui sont vraies dans le modèle standard, ces axiomes sont donc tous universellement réalisés par le λ_c-terme $\mathbf{I} \equiv \lambda z\,.\,z$. Par ailleurs, on vérifie sans difficulté que :

Proposition 4.4.10 (Réalisation des axiomes 3 et 4 de Peano)**.**

$$\lambda z\,.\,z\,\mathbf{I} \Vdash \forall x\, \neg(s(x) = 0),$$
$$\lambda z\,.\,z \Vdash \forall x\, \forall y\, (s(x) = s(y) \Rightarrow x = y).$$

En combinant les réalisateurs universels précédents avec le théorème d'adéquation (Théorème 4.4.6), il est alors clair que :

Théorème 4.4.11 (Réalisation des théorèmes de PA2)**.** — *Tout théorème* A *de* PA2 *est universellement réalisé par un* λ_c*-terme clos* $t : t \Vdash A$.

Preuve. Si A est un théorème de PA2, alors il existe des axiomes $A_1, \ldots, A_n$ de PA2 et un terme de preuve t tel que le jugement

$$z_1 : A_1, \ldots, z_n : A_n \vdash t : A$$

soit dérivable dans le système λNK2. En « branchant » des réalisateurs universels $u_1, \ldots, u_n$ des axiomes $A_1, \ldots, A_n$ sur les λ-variables $z_1, \ldots, z_n$, on obtient alors un réalisateur universel $t\{z_1 := u_1, \ldots, z_n := u_n\} \Vdash A$. ■

Le lecteur est invité à vérifier que les réalisateurs universels construits à l'aide des Prop. 4.4.9, 4.4.10 et du théorème ci-dessus n'utilisent que l'instruction call/cc, et sont dépourvus de toute constante de continuation k_π

(voir note 19 p. 103). Ainsi, les seules constantes de continuation k_π susceptibles d'apparaître au cours de l'évaluation d'un réalisateur universel construit à l'aide du Théorème 4.4.11 correspondent effectivement à des piles fournies par l'opposant au cours de l'interaction.

4.5 Extraction de témoin en réalisabilité classique

4.5.1 Le problème de l'extraction de témoin

Dans cette section, on s'intéresse au problème suivant : étant donné un théorème de PA2 de la forme $(\exists x \in \mathbf{N})\, A(x)$, comment extraire à partir de la démonstration de ce théorème un témoin existentiel, c'est-à-dire un entier naturel $n \in \mathbb{N}$ satisfaisant effectivement la propriété $A(n)$?

Avant d'aller plus loin, il est important de rappeler que dans le cadre d'une théorie classique telle que PA2, ce problème est indécidable dans sa forme la plus générale. En effet, la loi du tiers-exclu permet de dériver [21] pour chaque formule close C la formule existentielle

$$(\exists x \in \mathbf{N})\, ((x = 1 \wedge C) \vee (x = 0 \wedge \neg C))\,,$$

et il est clair que toute procédure permettant d'extraire un témoin existentiel (ici : un booléen indiquant la valeur de vérité de C) à partir de la démonstration de la formule ci-dessus nous donnerait aussitôt un algorithme de décision pour la théorie considérée, lequel n'existe pas [22].

Pour remédier à ce problème, la solution « historique » consiste à renoncer aux principe du tiers-exclu (qui est à la base du contre-exemple ci-dessus) et à se restreindre à la logique intuitionniste [23]. C'est dans ce cadre qu'a été développée la théorie de la *réalisabilité intuitionniste* [68, 70], qui permet d'extraire un témoin existentiel fiable à partir de n'importe quelle preuve *intuitionniste* d'une formule de la forme $(\exists x \in \mathbf{N})\, A(x)$.

Une autre solution, que nous allons présenter ici, consiste à conserver toute l'expressivité du raisonnement classique, mais à restreindre l'espace des formules existentielles candidates à l'extraction de témoin, en ne conservant que celles dont le corps $A(x)$ est une formule décidable. En pratique,

21. Dans le système λNK2, une démonstration de cette formule est donnée par le terme de preuve $\mathfrak{cc}\,(\lambda k\,.\,\langle \overline{0}, \mathsf{right}\,\langle \mathbf{I}, \lambda x\,.\,k\,\langle \overline{1}, \mathsf{left}\,\langle \mathbf{I}, x\rangle\rangle\rangle\rangle)$ (en notant $\langle t, u\rangle \equiv \lambda z\,.\,z\,t\,u$, où z est une λ-variable fraîche). On notera que ce terme de preuve ne dépend même pas de la formule C, ce qui rend d'autant plus vaine toute tentative d'extraire un témoin existentiel ici.

22. On sait en effet d'après le premier théorème d'incomplétude de Gödel que toute théorie cohérente, récursivement énumérable et contenant l'arithmétique de Peano (ce qui est le cas de PA2 à travers la relativisation $A \mapsto A^{\mathbf{N}}$) est incomplète et indécidable.

23. Ce qui, dans le λ_c-calcul, revient à renoncer à l'instruction $\mathfrak{cc}$ et aux constantes de continuation k_π (les « coups de bluff ») que cette instruction permet d'engendrer.

on ne considérera donc le problème de l'extraction de témoin que pour les formules de classe Σ^0_1, c'est-à-dire de la forme $(\exists x \in \mathbf{N})\, f(x) = 0$, où f est une fonction récursive primitive quelconque [24].

4.5.2 Mise en mémoire

Le premier problème à résoudre concerne la représentation des entiers naturels. En effet, nous avons vu à la Section 4.3.4 que chaque entier naturel $n \in \mathbb{N}$ est représenté dans le λ_c-calcul par l'*entier de Krivine*

$$\overline{n} :\equiv \overline{s}^n\overline{0} \equiv \underbrace{\overline{s}\,(\cdots(\overline{s}}_{n}\,\overline{0})\cdots), \qquad n \in \mathbf{N},$$

(avec $\overline{0} \equiv \lambda zf\,.\,z$ et $\overline{s} \equiv \lambda nzf\,.\,f\,(n\,z\,f)$), qu'on a coutume de considérer comme le terme de preuve (et réalisateur universel) « canonique » de la formule $n \in \mathbf{N}$ — c'est-à-dire comme une *valeur*.

Malheureusement, les réalisateurs de la formule $n \in \mathbf{N}$ sont loin d'avoir une forme aussi agréable, qui permet immédiatement d'identifier l'entier naturel $n \in \mathbb{N}$ dont il est question. Pour résoudre ce problème, il nous faut donc un dispositif capable d'*évaluer* un réalisateur de la formule $n \in \mathbf{N}$ afin de lui donner la forme voulue.

Pour ce faire, on étend le langage des formules (Section 4.2.3) avec une *implication en appel par valeur* notée $\{e\} \Rightarrow B$

Formules $\qquad A, B \quad ::= \quad \cdots \quad | \quad \{e\} \Rightarrow B\,,$

dont la valeur de fausseté (dans une valuation ρ) est définie par

$$\|(\{e\} \Rightarrow B)[\rho]\| = \{\overline{n} \cdot \pi \mid n = e[\rho]^{\mathbb{N}}, \pi \in \|B[\rho]\|\}.$$

Intuitivement, la formule $\{e\} \Rightarrow B$ représente le type des fonctions prenant en argument la représentation de l'expression arithmétique e sous la forme d'un entier de Krivine (c'est-à-dire sous forme de *valeur*), et retournant un objet de type B. Sémantiquement, il s'agit d'une construction très proche de l'implication $e \in \mathbf{N} \Rightarrow B$, à ceci près que cette dernière accepte tout réalisateur de la formule $e \in \mathbf{N}$, pas nécessairement évalué. Du point de vue de la réalisabilité, ces deux constructions sont équivalentes, car :

24. On notera que cette restriction sur la formule existentielle ne concerne que la *conclusion* de la démonstration. Le corps de la démonstration peut utiliser toutes les formes de quantification existentielle (du premier ou du second ordre) sans restriction, comme nous le verrons dans l'exemple de la Section 4.5.4.

Proposition 4.5.1 (Mise en mémoire)**.**

$$\lambda z\,.\,z \Vdash \forall x\,\forall Z\,((x \in \mathbf{N} \Rightarrow Z) \Rightarrow (\{x\} \Rightarrow Z)),$$
$$M \Vdash \forall x\,\forall Z\,((\{x\} \Rightarrow Z) \Rightarrow (x \in \mathbf{N} \Rightarrow Z)),$$

avec $M \equiv \lambda f n\,.\,n\,f\,(\lambda h x\,.\,h\,(\overline{s}\,x))\,\overline{0}$ *(« opérateur de mise en mémoire »).*

On voit que la réalisation de l'implication $(e \in \mathbf{N} \Rightarrow B) \Rightarrow (\{e\} \Rightarrow B)$ ne pose aucun problème, car le membre gauche est un sous-type du membre droit[25]. En revanche, la réalisation de l'implication réciproque nécessite un *opérateur de mise en mémoire*[26] M dont la fonction est d'évaluer le réalisateur de la formule $e \in \mathbf{N}$ (deuxième argument de M) avant de le passer à la fonction de type $\{e\} \Rightarrow B$ (premier argument)[27]. Sans surprise, cet opérateur sera un des ingrédients de la procédure d'extraction de témoin que nous allons maintenant présenter.

4.5.3 Extraction d'un témoin d'une formule Σ^0_1

À partir de maintenant, on suppose donnés une fonction récursive primitive f d'arité 1 et un réalisateur universel

$$t_0 \Vdash (\exists x \in \mathbf{N})\,f(x) = 0,$$

par exemple : un réalisateur universel construit à l'aide du Théorème 4.4.11 à partir d'une preuve de la formule $(\exists x \in \mathbf{N})\,f(x) = 0$ dans PA2.

On notera que dans l'absolu, la réalisabilité n'est pas nécessaire pour extraire un entier $n \in \mathbb{N}$ tel que $f(n) = 0$, puisqu'il suffit de tester successivement l'égalité $f(n) = 0$ sur tous les entiers, et de retourner la première solution rencontrée. Cependant, il s'agit d'une méthode très inefficace quand la première solution est très grande, et nous allons voir que l'exploitation du réalisateur universel t_0 permet de faire bien mieux.

L'opérateur d'extraction En pratique, l'extraction de témoin s'effectue en appliquant le réalisateur universel t_0 à l'*opérateur d'extraction de témoin*

$$E \;:\equiv\; M\,(\lambda x y\,.\,y\,(\mathsf{stop}\,x)),$$

25. On a en effet l'inclusion $\|\{e\} \Rightarrow B\| \subseteq \|e \in \mathbf{N} \Rightarrow B\|$ au niveau des valeurs de fausseté, d'où l'inclusion symétrique au niveau des valeurs de vérité.

26. La terminologie est sans doute malheureuse, mais c'est la terminologie consacrée.

27. La mise en mémoire est essentiellement une astuce calculatoire permettant de simuler l'*appel par valeur* dans la KAM, qui est fondamentalement en *appel par nom*.

où M est l'opérateur de mise en mémoire de la Prop. 4.5.1, et stop une instruction inerte [28] dont la seule fonction est de stopper la KAM.

On démontre alors que :

Théorème 4.5.2 (Extraction de témoin)**.** — *Si t_0 est un réalisateur universel de la formule $(\exists x \in \mathbf{N})\, f(x) = 0$, alors :*

$$t_0\, E \star \diamond \quad \succ \quad \mathsf{stop} \star \overline{n} \cdot \diamond,$$

où $n \in \mathbb{N}$ est un entier tel que $f(n) = 0$.

Preuve. Soit $E_0 \equiv \lambda xy\,.\,y\,(\mathsf{stop}\, x)$, de sorte que $E \equiv M\, E_0$. Pour montrer que le processus $t_0\, E \star \diamond$ s'évalue vers $\mathsf{stop} \star \overline{n} \cdot \diamond$ pour une certaine solution n de l'équation $f(n) = 0$, on se place dans le pôle $\perp\!\!\!\perp$ défini par

$$\perp\!\!\!\perp := \{p \in \Lambda \star \Pi \mid \exists n \in \mathbb{N}, f(n) = 0 \text{ et } p \succ \mathsf{stop} \star \overline{n} \cdot \diamond\}.$$

(Par construction, cet ensemble est clos par anti-évaluation.) Comme le terme t_0 réalise la formule $(\exists x \in \mathbf{N})\, f(x) = 0$ dans tout pôle, il la réalise en particulier dans le pôle $\perp\!\!\!\perp$ défini ci-dessus, d'où l'on tire que

$$t_0 \Vdash \forall x\, (x \in \mathbf{N} \Rightarrow f(x) = 0 \Rightarrow \dot{S}) \Rightarrow \dot{S}$$

en instanciant la variable Z par la valeur de fausseté $S := \{\diamond\}$.

Montrons à présent que $E_0 \Vdash \forall x\, (\{x\} \Rightarrow f(x) = 0 \Rightarrow \dot{S})$. Pour cela, il suffit de démontrer que pour toute pile $\pi \in \|\forall x\, (\{x\} \Rightarrow f(x) = 0 \Rightarrow \dot{S})\|$, on a $E_0 \star \pi \in \perp\!\!\!\perp$. D'après la Définition 4.4.2, une telle pile est de la forme $\pi \equiv \overline{n} \cdot v \cdot \diamond$, avec $n \in \mathbb{N}$ et $v \in |f(n) = 0|$. En évaluant $E_0 \star \pi$, il vient

$$E_0 \star \pi \;\equiv\; \lambda xy\,.\,y\,(\mathsf{stop}\, x) \star \overline{n} \cdot v \cdot \diamond \;\succ\; v \star \mathsf{stop}\, \overline{n} \cdot \diamond,$$

et comme le pôle $\perp\!\!\!\perp$ est clos par anti-évaluation, il suffit de montrer que $v \star \mathsf{stop}\, \overline{n} \cdot \diamond \in \perp\!\!\!\perp$ pour conclure que $E_0 \star \pi \in \perp\!\!\!\perp$. On distingue deux cas :

— Cas où l'égalité $f(n) = 0$ est vraie dans le modèle standard. D'après la définition du pôle, on a $\mathsf{stop}\, \overline{n} \star \diamond \succ \mathsf{stop} \star \overline{n} \cdot \diamond \in \perp\!\!\!\perp$, d'où il ressort que $\mathsf{stop}\, \overline{n} \Vdash \dot{S}$ d'après la définition de S. Par conséquent, on a $\mathsf{stop}\, \overline{n} \cdot \diamond \in \|\dot{S} \Rightarrow \dot{S}\| \subseteq \|f(n) = 0\|$ d'après le Lemme 4.4.8, et comme $v \in |f(n) = 0|$, il vient $v \star \mathsf{stop}\, \overline{n} \cdot \diamond \in \perp\!\!\!\perp$.

— Cas où l'égalité $f(n) = 0$ est fausse dans le modèle standard. Dans ce cas, on observe que $\mathsf{stop}\, \overline{n} \cdot \diamond \in \Lambda \cdot \Pi = \|f(n) = 0\|$ d'après le Lemme 4.4.8, d'où l'on tire à nouveau que $v \star \mathsf{stop}\, \overline{n} \cdot \diamond \in \perp\!\!\!\perp$.

28. C'est-à-dire une instruction ($\in \mathcal{K}$) sans règle d'évaluation.

Ce qui achève de montrer que $E_0 \Vdash \forall x\,(\{x\} \Rightarrow f(x) = 0 \Rightarrow \dot{S})$.

D'après la Prop. 4.5.1, on a $M \Vdash \forall x\,\forall Z\,((\{x\} \Rightarrow Z) \Rightarrow (x \in \mathbf{N} \Rightarrow Z))$. Par un calcul immédiat sur les valeurs de fausseté, on observe que

$$\begin{aligned}\big\|\forall x\,(\{x\} \Rightarrow f(x) = 0 \Rightarrow \dot{S}) \Rightarrow \forall x\,(x \in \mathbf{N} \Rightarrow f(x) = 0 \Rightarrow \dot{S})\big\| \\ \subseteq \big\|\forall x\,\forall Z\,((\{x\} \Rightarrow Z) \Rightarrow (x \in \mathbf{N} \Rightarrow Z))\big\|,\end{aligned}$$

d'où l'on tire (en utilisant l'inclusion inverse sur les valeurs de vérité) que

$$M \Vdash \forall x\,(\{x\} \Rightarrow f(x) = 0 \Rightarrow \dot{S}) \Rightarrow \forall x\,(x \in \mathbf{N} \Rightarrow f(x) = 0 \Rightarrow \dot{S}).$$

Par application, il vient alors $E \equiv M\,E_0 \Vdash \forall x\,(x \in \mathbf{N} \Rightarrow f(x) = 0 \Rightarrow \dot{S})$, et finalement $t_0\,E \Vdash \dot{S}$. Par conséquent, on a $t_0\,E \star \diamond \in \perp\!\!\!\perp$ (car $\diamond \in S$), ce qui implique que le processus $t_0\,E \star \diamond$ a le comportement calculatoire désiré, d'après la définition même du pôle $\perp\!\!\!\perp$. ■

Anatomie du processus d'extraction Le programme chargé d'extraire le témoin est construit en appliquant le réalisateur universel t_0 à l'opérateur d'extraction $E \equiv M\,(\lambda xy\,.\,y\,(\mathsf{stop}\,x))$, dont la fonction est de récupérer dans les λ-variables x et y les réalisateurs des formules $n \in \mathbf{N}$ et $f(n) = 0$ cachées dans le réalisateur t_0 (pour un certain entier naturel $n \in \mathbb{N}$ qu'il s'agit de déterminer). L'opérateur de mise en mémoire M garantit que le réalisateur de la formule $n \in \mathbf{N}$ est évalué avant d'être passé à la λ-variable x, de sorte que celle-ci ne reçoit pas un réalisateur arbitraire de la formule $n \in \mathbf{N}$, mais l'entier de Krivine $\overline{n}$ associé à l'entier naturel $n \in \mathbb{N}$ (ainsi déterminé).

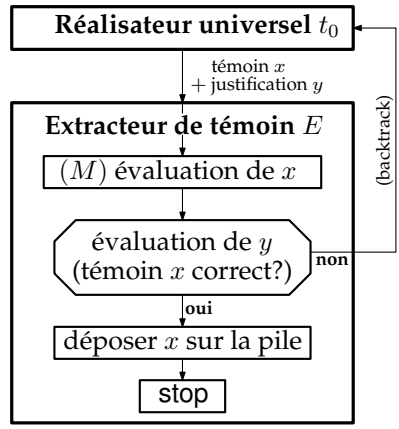
Une fois extraits l'entier de Krivine $\overline{n}$ et le réalisateur de l'égalité $f(n) = 0$, il s'agit de déterminer si le témoin proposé est correct ou non. Rappelons qu'en réalisabilité classique, un réalisateur de l'égalité $f(n) = 0$ ne garantit en rien que l'égalité soit vraie, car un tel réalisateur peut contenir des continuations cachées issues d'un « coup de bluff » du réalisateur t_0, auquel cas le témoin $\overline{n}$ proposé est un *faux témoin* [29]. Aussi, pour déterminer si le témoin est correct, on évalue le réalisateur de l'égalité $f(n) = 0$ (la *justification*) dans la construction $y\,(\mathsf{stop}\,x)$, en s'appuyant sur la forme particulière des réalisateurs de l'égalité (Lemme 4.4.8). En effet :

— Lorsque l'égalité $f(n) = 0$ est vraie, le réalisateur stocké dans y (de type $\forall Z\,(Z \Rightarrow Z)$) se comporte comme la fonction identité ; il donne donc le contrôle à son argument ($\mathsf{stop}\,x$), lequel dépose le témoin (correct) sur la pile et met fin à l'exécution.

29. L'existence de faux témoins est une spécificité de la réalisabilité classique, car en réalisabilité intuitionniste [68, 70], tous les témoins sont corrects par construction. Cette incertitude sur les témoins existentiels est le prix à payer pour étendre l'interprétation calculatoire des preuves à la logique classique.

— Lorsque l'égalité $f(n) = 0$ est fausse, le réalisateur stocké dans y (de type $\top \Rightarrow \bot$) ignore son argument et initie un backtrack, en utilisant une continuation cachée produite par le réalisateur t_0, qui reprend alors la main pour construire un nouveau témoin.

Ainsi, grâce aux continuations cachées dans les réalisateurs d'égalité, l'évaluation du réalisateur stocké dans y crée une *boucle de rétroaction* entre l'opérateur d'extraction E (chargé d'extraire et de contrôler les témoins produits par t_0) et le réalisateur universel t_0 (chargé de produire les témoins et les réalisateurs justifiant l'égalité).



De manière remarquable, ce processus de « négociation » entre le réalisateur universel t_0 et l'extracteur E permet de produire un témoin correct en temps fini, comme le démontre le Théorème 4.5.2.

On notera que la démonstration du Théorème 4.5.2 consiste à se placer dans un modèle de réalisabilité $\mathcal{M}_{\perp\!\!\!\perp}$ particulier, dont le pôle $\perp\!\!\!\perp$ a été spécialement construit pour capturer la propriété désirée (à savoir : l'ensemble des processus s'évaluant en temps fini vers une réponse correcte). Ce qui permet de propager automatiquement tous les invariants liés au problème considéré à travers l'interprétation de la Définition 4.4.2, et d'aboutir à la conclusion recherchée par des moyens purement sémantiques.

4.5.4 Exemple : le principe du minimum

Afin d'illustrer le fonctionnement de l'extracteur de témoin, nous allons considérer un exemple basé sur le *principe du minimum*, qui exprime que toute fonction des entiers dans les entiers atteint son minimum :

$$(\mathsf{PM}) \qquad (\forall x \in \mathbf{N})\, f(x) \in \mathbf{N} \Longrightarrow (\exists x \in \mathbf{N})\, (\forall y \in \mathbf{N})\, f(x) \leqslant f(y).$$

(L'ordre sur les entiers étant défini ici par la formule $x \leqslant y :\equiv x - y = 0$, où $x - y$ désigne la soustraction tronquée.)

La formule ci-dessus admet une preuve non constructive dans le système PA2 (elle se démontre par l'absurde en utilisant une induction forte). D'après le théorème 4.4.11, cette formule est donc universellement réalisée par un λ_c-terme clos PM, donné dans la Table 4.4 [30].

30. Pour des raisons de place, on a donné ici un réalisateur construit à la main. Un réalisateur universel extrait directement à partir d'une preuve dans PA2 serait beaucoup plus gros, mais aurait exactement le même comportement calculatoire.

$\mathbf{I}$	$:\equiv$	$\lambda x\,.\,x \qquad \mathbf{T} :\equiv \lambda xy\,.\,x \qquad \mathbf{F} :\equiv \lambda xy\,.\,y$
$\langle t_1, t_2\rangle$	$:\equiv$	$\lambda z\,.\,z\,t_1\,t_2$ (z λ-variable fraîche)
pred	$:\equiv$	$\lambda n\,.\,n\,\langle\overline{0},\overline{0}\rangle\,(\lambda p\,.\,p\,(\lambda xy\,.\,\langle x, \overline{s}\,x\rangle))\,(\lambda xy\,.\,x)$
minus	$:\equiv$	$\lambda n, m\,.\,m\,n\,\mathsf{pred}$
cmp	$:\equiv$	$\lambda n, m\,.\,\mathsf{minus}\,n\,m\,\mathbf{T}\,(\lambda\_\,.\,\mathbf{F})$
$\mathbf{Y}$	$:\equiv$	$(\lambda yf\,.\,f\,(y\,y\,f))\,(\lambda yf\,.\,f\,(y\,y\,f))$
PM	$:\equiv$	$\lambda f\,.\,\mathfrak{cc}\,(\lambda k\,.\,\mathbf{Y}\,(\lambda r, n\,.\,\langle n, \lambda m\,.\,\mathsf{cmp}\,(f\,n)\,(f\,m)\,\mathbf{I}\,(k\,(r\,m))\rangle)\,\overline{0})$
	$\Vdash$	$(\forall x \in \mathbf{N})\,f(x) \in \mathbf{N} \;\Rightarrow\; (\exists x \in \mathbf{N})(\forall y \in \mathbf{N})\,f(x) \leqslant f(y)$

TABLE 4.4 – *Réalisation du principe du minimum*

On notera que dans le système PA2, le principe du minimum est conditionné par la formule $\mathsf{Total}(f) \equiv (\forall x \in \mathbf{N})\,f(x) \in \mathbf{N}$ exprimant la totalité de la fonction f considérée [31]. Heureusement, nous avons vu à la Prop. 4.2.2 p. 91 que cette hypothèse admet une démonstration en logique du second ordre pour chaque fonction récursive primitive f [32], dont le terme de preuve est noté tot_f $(: \mathsf{Total}(f))$ dans ce qui suit.

Intuitivement, on peut voir la démonstration non constructive du principe du minimum comme un « oracle » associant à chaque fonction f (satisfaisant l'hypothèse $\mathsf{Total}(f)$) l'entier naturel où cette fonction atteint son minimum. Bien entendu, il n'est pas question d'implémenter un tel oracle, même en λ_c-calcul ! En revanche, nous allons voir que le λ_c-terme PM (Table 4.4) *simule* un tel oracle à l'aide du mécanisme d'essai/erreur offert par l'instruction $\mathfrak{cc}$ et les constantes de continuation.

Une conséquence Σ^0_1 du principe du minimum Il n'est pas possible d'appliquer directement la technique d'extraction de témoin au réalisateur universel du principe du minimum, car la conclusion n'est pas une formule de classe Σ^0_1. En revanche, on remarque que pour toute fonction récursive primitive g d'arité 1, le principe du minimum implique trivialement l'existence d'un entier naturel x tel que $f(x) \leqslant f(g(x))$, en considérant précisément le point x où la fonction f atteint son minimum. Dans le système λNK2, cet

31. Rappelons que dans le système PA2, qui ne comporte pas de schéma d'induction, les individus ne sont pas tous des entiers *a priori*. Dans la formalisation, il est donc vital d'expliciter quels individus sont des entiers (à l'aide du prédicat $\mathbf{N}$) et quelles fonctions envoient les entiers dans les entiers (à l'aide de la formule $\mathsf{Total}(f)$).

32. On peut par ailleurs démontrer (ce que nous ne ferons pas ici) qu'un réalisateur universel de la formule $\mathsf{Total}(f)$ est un λ_c-terme calculant la fonction f.

argument très simple se formalise de la manière suivante :

$$\begin{array}{l} h \;:\; \overbrace{\mathsf{Total}(f) \;\Rightarrow\; (\exists x \in \mathbf{N})(\forall y \in \mathbf{N})\, f(x) \leqslant f(y)}^{\text{principe du minimum}} \quad \vdash \\ \quad h\,(\mathsf{tot}_f)\,(\lambda x, z\,.\,\langle x, z\,(\mathsf{tot}_g\, x)\rangle) \quad : \quad \underbrace{(\exists x \in \mathbf{N})\, f(x) \leqslant f(g(x))}_{\text{conséquence } \Sigma^0_1}. \end{array}$$

Ainsi, en substituant la λ-variable h par le λ_c-terme PM (Théorème 4.4.6), on obtient le réalisateur universel suivant :

$$\begin{array}{rl} t_0 :\equiv & \mathsf{PM}\,(\mathsf{tot}_f)\,(\lambda x, z\,.\,\langle x, z\,(\mathsf{tot}_g\, x)\rangle) \\ \Vdash & (\exists x \in \mathbf{N})\, f(x) \leqslant f(g(x)). \end{array}$$

La formule réalisée étant de classe Σ^0_1, on sait alors (Théorème 4.5.2) que le processus $t_0\, E \star \diamond$ s'évalue en temps fini vers un processus terminal de la forme $\mathsf{stop} \star \overline{n} \cdot \diamond$, où n est un entier naturel tel que $f(n) \leqslant f(g(n))$.

Exécution du programme $E\, t_0$ On suppose à présent que les fonctions f et g sont données par $f(n) = |n - 1000|$ (minimum atteint pour $n = 1000$) et $g(n) = 2n + 1$. Afin d'afficher en cours d'évaluation les témoins intermédiaires proposés par le réalisateur universel t_0, on utilise une version modifiée de l'opérateur d'extraction E donnée par

$$E := M\,(\lambda xy\,.\,\mathsf{print}\, x\, y\,(\mathsf{stop}\, x)),$$

où « print » est une instruction affichant son premier argument avant de donner le contrôle à l'argument suivant. (Formellement, elle est donc spécifiée par la règle d'évaluation $\mathsf{print} \star u \cdot v \cdot \pi \;\succ\; v \star \pi$, l'affichage du terme u constituant la partie informelle de la spécification.)

La trace d'exécution du processus $t_0\, E \star \diamond$ nous donne alors :

Témoins affichés :	$\overline{0},\ \overline{1},\ \overline{3},\ \overline{7},\ \overline{15},\ \overline{31},\ \overline{63},\ \overline{127},\ \overline{255},\ \overline{511},\ \overline{1023}$
État final :	$\mathsf{stop} \star \overline{1023} \cdot \diamond$

Ainsi, l'extraction aboutit au témoin existentiel $n = 1023$ (qui satisfait effectivement la formule $f(n) \leqslant f(g(n))$) en seulement 11 itérations.

En insérant des instructions « print » supplémentaires dans le code, il est facile de reconstituer le mécanisme de négociation qui a lieu entre les réalisateurs PM, t_0 et l'extracteur E au cours de l'évaluation :

1. À chaque itération, le pseudo-oracle PM produit un témoin n_i (valeur initiale : $n_0 = 0$) et un réalisateur de $(\forall y \in \mathbf{N}) f(n_i) \leqslant f(y)$. Ce réalisateur « bluffe » en embarquant une continuation cachée, qui permettra à PM de se dédire en cas d'erreur.

2. Le réalisateur universel t_0 reprend le témoin n_i produit par PM, et transforme le réalisateur qui l'accompagne en un réalisateur de l'égalité $f(n_i) \leqslant f(g(n_i))$ $(\equiv f(n_i) - f(g(n_i)) = 0)$.
3. L'extracteur E récupère le témoin n_i et le réalisateur de l'égalité fournis par t_0, et évalue le réalisateur de l'égalité.
   — Lorsque $f(n_i) \leqslant f(g(n_i))$, l'extracteur E garde le contrôle : il dépose le témoin n_i sur la pile et stoppe l'exécution.
   — Lorsque $f(n_i) > f(g(n_i))$, la continuation cachée dans le réalisateur de l'égalité redonne le contrôle au pseudo-oracle PM (retour à l'étape 1). Celui-ci utilise alors le contre-exemple pour proposer un nouveau témoin $n_{i+1} := g(n_i)$ [33].

On notera qu'à aucun moment le pseudo-oracle PM n'est en mesure de produire le point $(n = 1000)$ où la fonction f atteint son minimum absolu. Cependant, le dernier témoin $(n_{11} = 1023)$ est une approximation suffisamment bonne pour constituer une solution du problème $f(n) \leqslant f(g(n))$.

33. On notera que le réalisateur t_0 a accès aux fonctions f et g à travers les « sous-programmes » $\mathsf{tot}_f \Vdash (\forall x \in \mathbf{N})\, f(x) \in \mathbf{N}$ et $\mathsf{tot}_g \Vdash (\forall x \in \mathbf{N})\, g(x) \in \mathbf{N}$, dont on peut démontrer qu'ils calculent effectivement les fonctions f et g.

Chapitre 5

Analyses d'erreur en arithmétique flottante

Claude-Pierre Jeannerod
Nathalie Revol

L'arithmétique à virgule flottante est le moyen le plus couramment utilisé pour calculer avec des (approximations des) réels sur ordinateur. Si cette arithmétique est par nature inexacte, la norme IEEE 754 qui la régit définit un cadre strict dans lequel il est possible d'analyser et prédire de façon tout à fait rigoureuse le comportement de nombreux algorithmes numériques de base. Dans ce cours, nous illustrerons ce principe à l'aide de résultats obtenus très récemment, qui revisitent une partie de l'analyse numérique classique. Nous verrons aussi comment majorer de façon automatique les erreurs dues à l'arithmétique flottante et présenterons quelques résultats expérimentaux : l'approche présentée utilise l'arithmétique par intervalles. Comme l'arithmétique par intervalles est implantée en utilisant l'arithmétique flottante, nous détaillerons comment l'implantation tient compte des erreurs qui se produisent dans ses calculs internes.

5.1 Introduction

Précision finie et qualité numérique. L'arithmétique à virgule flottante est le moyen le plus couramment utilisé pour calculer avec des réels sur ordinateur, et ce notamment grâce à la spécification rigoureuse fournie par la norme IEEE 754 [48]. Du fait que cette arithmétique n'utilise qu'une précision finie (c'est-à-dire, un nombre de chiffres fixé à l'avance), chaque opération élémentaire comme $+$ ou $\times$ est susceptible de commettre une

petite erreur, appelée *erreur d'arrondi*. De plus, lors d'un calcul impliquant plusieurs opérations élémentaires, ces erreurs auront en général tendance à s'accumuler ou l'une d'entre elles pourra être amplifiée. Une question importante est donc de comprendre l'effet de ces erreurs sur la *qualité* du résultat renvoyé.

Dans ce cours, nous verrons deux façons d'aborder cette question, l'analyse dite *a priori* et une analyse automatique, *a posteriori*, et nous donnerons un aperçu des évolutions récentes de chacun de ces deux domaines.

Analyse a priori. Cette approche vise à analyser la façon dont un algorithme se comporte en arithmétique à virgule flottante, et ce quelles que soient les valeurs des données en entrée. Idéalement, on cherche à garantir a priori que la solution calculée $\widehat{s}$ a une certaine qualité numérique. Cela peut signifier que $\widehat{s}$ est « proche » d'une solution exacte ou, ce qui peut s'avérer plus facile à établir, que $\widehat{s}$ est la solution exacte pour des données « proches ». Dans les deux cas, il s'agit d'établir des majorations fines et rigoureuses de l'erreur commise. Dans ce cours, l'accent sera mis sur le rôle joué par l'arithmétique dans l'obtention de telles bornes. Nous verrons en particulier comment exploiter les (nombreuses) propriétés des nombres flottants pour établir des bornes fines, concises et faciles à interpréter.

Analyse automatique, a posteriori. Ici, le calcul de bornes d'erreur s'effectue en même temps que le calcul étudié, avec un jeu de valeurs pour les entrées. Ces entrées peuvent éventuellement être données avec une incertitude initiale ; l'analyse incorporera alors cette incertitude et les erreurs d'arrondi qui se produisent lors des calculs et produira un encadrement de l'erreur sur le résultat final. Le pire cas étant considéré pour chacune de ces erreurs prise indépendamment, l'encadrement obtenu est fréquemment trop pessimiste : nous verrons quelques pistes pour obvier à ce problème.

Plan du cours. La section 5.2 donne une présentation détaillée de l'arithmétique à virgule flottante. Nous insisterons surtout sur ses deux principaux ingrédients, l'ensemble des nombres flottants et la notion d'arrondi correct, ainsi que sur quelques unes des nombreuses propriétés qui en découlent.

Nous verrons ensuite en section 5.3 comment exploiter ces propriétés pour analyser a priori le comportement de divers algorithmes classiques. Après un rappel de l'approche la plus courante (l'analyse inverse de Wilkinson), nous verrons quelques améliorations récentes obtenues pour des

problèmes de base comme la somme de n nombres, le produit de matrices, ou encore l'évaluation très précise d'expressions de la forme $ab + cd$.

Nous donnerons une présentation générale de l'arithmétique par intervalles, qui est l'approche mise en œuvre dans ce cours pour analyser automatiquement les erreurs, en section 5.4, puis nous verrons les détails de son implantation utilisant l'arithmétique flottante. Nous l'avons déjà signalé, l'encadrement des erreurs peut être pessimiste. Une solution qui demande peu d'expertise consiste, sans rien modifier d'autre, à augmenter la précision de calcul. Nous montrerons les bénéfices d'un changement de représentation des intervalles, par leur centre et leur rayon plutôt que par leurs extrémités. Enfin, nous évoquerons deux variantes qui réduisent les problèmes provoqués par la « perte » de certaines propriétés algébriques.

Nous conclurons en section 5.5 avec des suggestions de lectures pour approfondir les notions et approches présentées ou pour les prolonger.

5.2 Arithmétique à virgule flottante

5.2.1 Nombres à virgule flottante

Informellement, il s'agit de nombres rationnels de la forme $M\beta^E$ où β est un entier positif fixé, appelé *base*, et où M et E sont deux entiers relatifs de taille bornée. Plus précisément, $|M| < \beta^p$ où p est un entier positif appelé *précision*, et $E + p - 1 \in [e_{\min}, e_{\max}]$ où $e_{\min}$ et $e_{\max}$ sont deux entiers définissant la *plage d'exposants*.

L'ensemble $\mathbb{F}$ de tels nombres, appelé *système à virgule flottante*, est donc un sous-ensemble très particulier de $\mathbb{Q}$, entièrement caractérisé par les paramètres entiers β, p, $e_{\min}$ et $e_{\max}$. Par exemple, parler de « double précision IEEE » revient à fixer $\beta = 2$, $p = 53$, $e_{\min} = -1022$ et $e_{\max} = 1023$.

L'objectif de ce cours étant d'étudier l'effet d'une précision finie sur la qualité des calculs, nous idéaliserons un peu en fixant $e_{\min} = -\infty$ et $e_{\max} = +\infty$ de façon à ne jamais devoir nous préoccuper d'éventuels dépassements de capacité au niveau de l'exposant. En particulier, l'absence de borne inférieure sur la valeur des exposants permet de représenter tout élément non nul de $\mathbb{F}$ de façon unique sous forme *normalisée*, c'est-à-dire avec une mantisse M telle que $\beta^{p-1} \leqslant |M|$. L'ensemble $\mathbb{F}$ des nombres à virgule flottante en base β et précision p pourra donc être décrit comme suit :

$$\mathbb{F} = \{0\} \cup \left\{ M\beta^E \mid M, E \in \mathbb{Z}, \beta^{p-1} \leqslant |M| < \beta^p \right\}. \tag{5.1}$$

Hypothèses. Dans la suite de ce cours, nous supposerons que β est pair et que $p \geqslant 2$. Ces hypothèses nous permettront de simplifier certains énoncés

et certaines preuves, sans pour autant exclure de cas pratique significatif. En particulier, la norme IEEE 754 suppose que $\beta \in \{2, 10\}$ et les formats classiques qu'elle spécifie (et qui sont ceux dont nous disposons aujourd'hui sur nos machines) vérifient tous $p \geqslant 11$.

Représentations alternatives. Selon les cas, il peut être utile de décrire les éléments de $\mathbb{F}$ autrement qu'avec (5.1). En posant $m = |M|/\beta^{p-1}$ et $e = E + p - 1$, on voit tout d'abord que tout $f \in \mathbb{F}_{\neq 0}$ est aussi de la forme

$$\pm m\beta^e, \qquad m = \sum_{i=0}^{p-1} m_i \beta^{-i} =: (m_0 \bullet m_1 \dots m_{p-1})_\beta \tag{5.2}$$

avec $m_i \in \{0, 1, \dots, \beta - 1\}$ pour tout i, $m_0 \neq 0$ et $e \in \mathbb{Z}$. Cette seconde représentation, bien qu'étant moins concise que la première (qui est exclusivement entière), a l'avantage de faire apparaître explicitement la virgule ainsi que les chiffres de la mantisse en base β ; en particulier, m_0 est appelé *chiffre le plus significatif* et m_{p-1} est appelé *chiffre le moins significatif*.

Une troisième représentation est obtenue en récrivant la mantisse fractionnaire m apparaissant dans (5.2) à l'aide de l'*unité d'erreur d'arrondi*

$$u = \frac{1}{2}\beta^{1-p}.$$

En effet, m prenant les valeurs $1, 1 + \beta^{1-p}, 1 + 2\beta^{1-p}, \dots$, on vérifie facilement que tout élément de $\mathbb{F}$ est de la forme

$$\pm (1 + 2ku)\beta^e \tag{5.3}$$

avec $k \in \{0, 1, \dots, (\beta - 1)\beta^{p-1} - 1\}$ et $e \in \mathbb{Z}$. Comme nous le verrons plus loin, l'unité d'erreur d'arrondi u joue un rôle central dans toute analyse a priori, et il peut s'avérer particulièrement pratique que cette quantité apparaisse d'emblée dans la façon d'écrire les éléments de $\mathbb{F}$.

Propriétés. L'ensemble $\mathbb{F}$ défini en (5.1) peut être vu comme une « grille de nombres structurés » qui, à ce titre, possède de nombreuses propriétés. Notons tout d'abord que si $f \in \mathbb{F}$ alors $-f \in \mathbb{F}$ (symétrie) et $f\beta^e \in \mathbb{F}$ pour tout $e \in \mathbb{Z}$ (auto-similarité). Par conséquent, nous serons souvent amenés à nous concentrer sur le sous-ensemble $\mathbb{F} \cap [1, \beta)$; en rappelant (5.3), nous voyons que ce sous-ensemble est de la forme

$$\mathbb{F} \cap [1, \beta) = \{1, 1 + 2u, 1 + 4u, \dots, \beta - 2u\}.$$

Plus généralement, tout sous-ensemble de la forme $\mathbb{F} \cap [\beta^e, \beta^{e+1}]$ avec $e \in \mathbb{Z}$ possède la double propriété suivante :

— il contient $(\beta - 1)\beta^{p-1} + 1$ éléments régulièrement espacés ;
— la distance entre deux éléments consécutifs est égale à

$$2u\beta^e. \tag{5.4}$$

Ce second point implique qu'une puissance entière de la base est plus éloignée (d'un facteur multiplicatif égal à β) de son successeur que de son prédécesseur dans $\mathbb{F}$. Par exemple, le voisinage de 1 a la forme suivante :

$$\ldots, 1 - \frac{4u}{\beta}, 1 - \frac{2u}{\beta}, 1, 1 + 2u, 1 + 4u, \ldots$$

Du fait de cette « régularité par morceaux » de l'ensemble $\mathbb{F}$, les analyses d'erreur seront parfois plus délicates au voisinage des puissances entières de la base.

Enfin, certaines propriétés de $\mathbb{F}$ s'expriment facilement à l'aide des notions d'ufp (*unit in the first place*) et d'ulp (*unit in the last place*), définies de la façon suivante : pour $f \in \mathbb{F}_{\neq 0}$ représenté par $f = \pm m\beta^e$ comme en (5.2),

$$\mathrm{ufp}(f) = \beta^e \quad \text{et} \quad \mathrm{ulp}(f) = \beta^{e-p+1}.$$

En d'autres termes, $\mathrm{ufp}(f)$ donne l'ordre de grandeur de f, et $\mathrm{ulp}(f)$ le « poids » de son chiffre le moins significatif. Tout comme les fonctions usuelles 'signe' et 'valeur absolue', ces fonctions ufp et ulp permettront de manipuler les éléments de $\mathbb{F}$ indépendamment de leur représentation [1]. Parmi les propriétés utiles pour cela, mentionnons le fait que ces deux fonctions sont croissantes, la relation $\mathrm{ulp}(f) = \mathrm{ufp}(f)\beta^{1-p}$, et l'implication

$$f \neq 0 \Rightarrow \mathrm{ulp}(f) \leqslant |f|/\beta^{p-1} < \beta\mathrm{ulp}(f). \tag{5.5}$$

De plus, en rappelant que les exposants E et e dans (5.1) et (5.2) vérifient $e = E + p - 1$, nous pouvons écrire $\mathrm{ulp}(f) = \beta^E$ et conclure en particulier que

$$f \in \mathrm{ulp}(f)\mathbb{Z}. \tag{5.6}$$

Cette propriété, qui signifie que *tout nombre à virgule flottante est un multiple entier de son ulp*, sera très utile pour analyser des situations de calcul exact, comme le théorème de Sterbenz ou certaines *transformations exactes* ; voir en section 5.2.5.

1. La fonction ufp a été introduite dans [119] et est en fait définie plus généralement sur les réels par $\mathrm{ufp}(0) = 0$ et $\mathrm{ufp}(t) = \beta^{\lfloor \log_\beta |t| \rfloor}$ pour tout $t \in \mathbb{R}_{\neq 0}$.

5.2.2 Fonctions d'arrondi

L'approximation d'un réel par un nombre flottant se fait à l'aide d'une fonction $\circ : \mathbb{R} \to \mathbb{F}$ appelée *fonction d'arrondi* et telle que

$$t \in \mathbb{F} \Rightarrow \circ(t) = t; \qquad t \leqslant t' \Rightarrow \circ(t) \leqslant \circ(t'). \tag{5.7}$$

Cette définition générale (5.7) suffit déjà à garantir les trois propriétés importantes suivantes, dont la vérification est immédiate :

$$\begin{gathered} \circ(t) = 0 \Leftrightarrow t = 0; \\ \beta^e \leqslant |t| < \beta^{e+1}, e \in \mathbb{Z} \Rightarrow \beta^e \leqslant |\circ(t)| \leqslant \beta^{e+1}; \\ \operatorname{sign}(\circ(t)) = \operatorname{sign}(t). \end{gathered}$$

En d'autres termes, arrondir préserve le signe et, à un facteur β près, l'ordre de grandeur.

Nous allons maintenant spécialiser cette définition générale de façon à définir différents types d'arrondi (« au plus proche », « vers zéro », etc.).

Arrondi « au plus proche ». C'est le type d'arrondi le plus utilisé en pratique ; il sera noté RN pour « *rounding to nearest* » et vérifie, par définition,

$$|t - \mathrm{RN}(t)| = \min_{f \in \mathbb{F}} |t - f| \quad \text{pour tout } t \in \mathbb{R}. \tag{5.8}$$

Cette propriété, qui implique (5.7), doit être complétée par une règle (« *tie-breaking rule* ») définissant $\mathrm{RN}(t)$ de façon unique pour chaque réel t situé à mi-chemin entre deux éléments de $\mathbb{F}$ consécutifs.

La règle la plus commune, spécifiée dès la première version de la norme IEEE 754 en 1985, consiste à choisir le nombre flottant dont la mantisse entière M est paire. Par exemple, cette règle impose d'arrondir $t = 1 + u$ vers 1 puisque les deux nombres flottants qui encadrent t sont $f_1 = 1$ et $f_2 = 1 + 2u$ et que leurs mantisses entières sont $M_1 = \beta^{p-1}$ et $M_2 = \beta^{p-1} + 1$. La révision de 2008 de la norme IEEE 754 a introduit une seconde règle, selon laquelle le flottant retourné est celui de plus grande valeur absolue. Avec cette seconde règle (destinée surtout au calcul financier en base 10), l'arrondi au plus proche de $1 + u$ n'est plus 1, mais $1 + 2u$.

Dans la suite de ce cours, nous n'aurons pas à nous restreindre à ces deux règles, propres à la norme IEEE 754 ; nous pourrons plus généralement considérer n'importe quelle règle « raisonnable » qui, comme celles-ci, ne dépend ni du signe ni de l'ordre de grandeur du réel à arrondir. Cela reviendra à supposer que, pour tout $t \in \mathbb{R}$, la fonction RN vérifie

$$\mathrm{RN}(-t) = -\mathrm{RN}(t) \quad \text{et} \quad \mathrm{RN}(t\beta^k) = \mathrm{RN}(t)\beta^k, \quad \forall k \in \mathbb{Z}.$$

Ces hypothèses reflètent bien la symétrie et la structure de l'ensemble $\mathbb{F}$ et pourront, elles aussi, être exploitées pour simplifier certaines analyses.

Arrondis dirigés. La norme IEEE 754 spécifie aussi trois sortes d'arrondi dirigé, que nous noterons RD, RU et RZ pour « *rounding down* », « *rounding up* » et « *rounding to zero* ». Ces fonctions sont définies à partir de (5.7) et de l'une des trois inégalités suivantes, valables pour tout $t \in \mathbb{R}$:

$$\mathrm{RD}(t) \leqslant t, \qquad t \leqslant \mathrm{RU}(t), \qquad |\mathrm{RZ}(t)| \leqslant |t|.$$

Cela signifie que $\mathrm{RD}(t)$ est le plus grand élément de $\mathbb{F}$ inférieur ou égal à t, que $\mathrm{RU}(t)$ est le plus petit élément de $\mathbb{F}$ supérieur ou égal à t, et que RZ tronque t pour n'en garder que les p chiffres (en base β) les plus significatifs.

Comme pour RN, nous avons $\circ(t\beta^k) = \circ(t)\beta^k$ pour tout $k \in \mathbb{Z}$ et $\circ \in \{\mathrm{RD}, \mathrm{RU}, \mathrm{RZ}\}$, ainsi que $\mathrm{RZ}(-t) = -\mathrm{RZ}(t)$. Cette dernière propriété n'est en revanche pas satisfaite par RU et RD, pour lesquels nous avons la relation $\mathrm{RD}(-t) = -\mathrm{RU}(t)$.

Notons enfin que si $t \notin \mathbb{F}$ alors $\mathrm{RD}(t)$ et $\mathrm{RU}(t)$ définissent l'unique paire d'éléments consécutifs de $\mathbb{F}$ telle que $t \in [\mathrm{RD}(t), \mathrm{RU}(t)]$.

Bornes d'erreur. Lorsque t n'est pas un flottant, lui appliquer une fonction d'arrondi $\circ$ entraîne systématiquement une erreur, appelée *erreur d'arrondi* et que l'on peut définir de façon absolue comme $|t - \circ(t)|$ ou de façon relative à l'aide des deux fonctions de $\mathbb{R}_{\neq 0}$ dans $\mathbb{R}$ suivantes :

$$E_1(t) = \frac{|t - \circ(t)|}{|t|} \qquad \text{et} \qquad E_2(t) = \frac{|t - \circ(t)|}{|\circ(t)|}.$$

Voyons maintenant comment majorer ces trois erreurs lorsque $\circ = \mathrm{RN}$. En notant e l'entier tel que $\beta^e \leqslant |t| < \beta^{e+1}$ et en rappelant que la distance entre deux éléments consécutifs de $\mathbb{F} \cap [\beta^e, \beta^{e+1}]$ vaut $2u\beta^e$, nous voyons que l'erreur absolue commise en arrondissant « au plus proche » vérifie

$$|t - \mathrm{RN}(t)| \leqslant \frac{1}{2} \times 2u\beta^e = u\beta^e. \tag{5.9}$$

Cette borne dépend de l'unité d'erreur d'arrondi u mais aussi de l'ordre de grandeur du réel t. Puisque $\beta^e \leqslant |t|$ implique $\beta^e \leqslant |\mathrm{RN}(t)|$, nous déduisons ensuite que $E_2(t) \leqslant u$. Cela vaut également pour $E_1(t)$, mais dans ce cas il est possible de faire légèrement mieux : en effet,

— si $|t| \geqslant (1+u)\beta^e$ alors $E_1(t) \leqslant u\beta^e/|t| \leqslant u/(1+u)$;
— si $|t| < (1+u)\beta^e$ alors $|\mathrm{RN}(t)| = \beta^e$ et, puisque $\mathrm{RN}(t)$ et t ont le même signe, $E_1(t) = 1 - \beta^e/|t| < 1 - 1/(1+u) = u/(1+u)$.

Nous venons donc de démontrer le résultat suivant :

Théorème 5.2.1. *Pour tout réel t non nul, les erreurs relatives commises en l'arrondissant « au plus proche » vérifient*

$$E_1(t) \leqslant \frac{u}{1+u} \qquad et \qquad E_2(t) \leqslant u.$$

L'inégalité $E_1(t) \leqslant u/(1+u)$ peut être attribuée à Dekker [29] et Knuth [69, p. 232], et l'inégalité $E_2(t) \leqslant u$ apparaît dans [47, p. 39]. Ces deux bornes sont atteintes par exemple pour $t = 1+u$ et avec la règle d'arrondi « au plus proche vers le flottant de mantisse paire ». Leur principal intérêt est de majorer l'erreur d'arrondi indépendamment de la valeur du réel à arrondir, par une quantité définie exclusivement par la base et la précision. Par exemple, en base 2 nous avons $u = 2^{-p}$ et donc

$$\frac{u}{1+u} \approx u \approx \begin{cases} 5.9 \times 10^{-8} & \text{si } p = 24 \text{ (simple précision),} \\ 1.1 \times 10^{-16} & \text{si } p = 53 \text{ (double précision).} \end{cases}$$

Pour les arrondis dirigés, il suffit essentiellement de remplacer u par $2u$ dans les trois bornes précédentes. Plus précisément, l'analogue de (5.9) pour $\circ \in \{\mathrm{RD}, \mathrm{RU}, \mathrm{RZ}\}$ est l'inégalité stricte $|t - \circ(t)| < 2u\beta^e$ et, concernant les erreurs relatives, on vérifie facilement que $E_1(t) < 2u/(1+2u)$ et $E_2(t) < 2u$ lorsque $\circ = \mathrm{RZ}$, et que $E_1(t), E_2(t) < 2u$ lorsque $\circ \in \{\mathrm{RD}, \mathrm{RU}\}$.

5.2.3 Arrondi correct des opérations élémentaires

Le résultat exact d'une opération élémentaire $\mathrm{op} \in \{+, -, \times, /\}$ sur des éléments de $\mathbb{F}$ n'est en général pas dans $\mathbb{F}$. Pour une fonction d'arrondi $\circ$ donnée, l'arithmétique à virgule flottante va alors consister à approcher le résultat exact $r = x \,\mathrm{op}\, y$ par la valeur $\widehat{r} \in \mathbb{F}$ de son arrondi :

$$\widehat{r} := \circ(x \,\mathrm{op}\, y). \tag{5.10}$$

Par exemple, si $x = y = \beta^p - 1$, le produit $xy = \beta^{2p} - 2\beta^p + 1$ est un entier ayant $2p$ chiffres en base β (soit le double de la précision p offerte par $\mathbb{F}$) :

$$xy = (\underbrace{* * * \ldots *}_{p-1} \bar{*} \underbrace{00 0 \ldots 0}_{p-1} 1)_\beta, \qquad * = \beta - 1 \neq 0, \qquad \bar{*} = \beta - 2.$$

Ainsi, $xy \notin \mathbb{F}$ et, en arithmétique à virgule flottante avec arrondi au plus proche, le résultat de cette multiplication est $\widehat{r} = \mathrm{RN}(xy) = \beta^{2p} - 2\beta^p = (\beta^p - 2)\beta^p \in \mathbb{F}$.

La méthode de calcul définie par (5.10) est appelée *arrondi correct* et peut être vue comme un algorithme, un mécanisme de composition de fonctions permettant de contrôler à chaque opération l'augmentation éventuelle de la taille du résultat exact [129]. La norme IEEE 754 exige l'arrondi correct pour les quatre opérations de base, la racine carrée et l'opération $(x, y, z) \mapsto xy + z$ (appelée FMA pour « *fused multiply-add* »). Dans tous les cas, $\circ \in \{\mathrm{RN}, \mathrm{RD}, \mathrm{RU}, \mathrm{RZ}\}$ avec, pour RN, les deux règles évoquées en section 5.2.2 pour départager les cas d'égalité.

Grâce à (5.10), le résultat d'une opération en virgule flottante est donc défini sans aucune ambiguïté. En pratique, la norme IEEE 754 spécifie aussi le résultat à retourner en cas de dépassement de capacité (la plage d'exposants étant bornée) ou en cas d'opération illicite (comme $0/0$ ou $\sqrt{-1}$), et ce grâce aux notions d'infinis et de NaN (*Not-a-Number*). Dans ce cours, nous nous contenterons d'exploiter (5.10), mais il faut garder à l'esprit que l'implantation efficace de cette règle ainsi que de tous les autres aspects de la norme IEEE 754 est un domaine de recherche en soi (représenté notamment par la série de conférences ARITH).

Disposer d'une arithmétique avec arrondis dirigés, et en particulier RD et RU, permet de calculer les deux flottants consécutifs qui encadrent le résultat exact (si celui-ci n'est pas lui-même un flottant) :

$$x \operatorname{op} y \in [\mathrm{RD}(x \operatorname{op} y), \mathrm{RU}(x \operatorname{op} y)].$$

Comme nous le verrons en section 5.4, cela sera très pratique pour implanter les arithmétiques par intervalles.

Modélisation standard. En appliquant à $t = x \operatorname{op} y$ les bornes d'erreur du théorème 5.2.1, nous voyons que pour l'arrondi au plus proche la valeur calculée $\widehat{r} = \mathrm{RN}(x \operatorname{op} y)$ est de la forme

$$\widehat{r} = (x \operatorname{op} y)(1 + \delta_1), \qquad |\delta_1| \leqslant \frac{u}{1+u}; \tag{5.11}$$

$$= \frac{x \operatorname{op} y}{1 + \delta_2}, \qquad |\delta_2| \leqslant u. \tag{5.12}$$

Pour les arrondis dirigés, la discussion en fin de section 5.2.2 montre qu'il suffit de remplacer les deux inégalités ci-dessus par $|\delta_1|, |\delta_2| < 2u$. (Dans tous les cas, δ_1 et δ_2 désignent des réels dont la valeur est inconnue a priori.)

Les relations (5.11) et (5.12) sont appelées *modèles standard de l'arithmétique à virgule flottante* et expriment la principale conséquence de l'arrondi correct : les erreurs relatives commises par chaque opération élémentaire sont majorées par une « petite constante » exprimable exclusivement à

l'aide de l'unité d'erreur d'arrondi u. En pratique, ces modèles sont valables pour l'arithmétique IEEE 754 dès lors que l'opération en question n'entraîne aucun dépassement de capacité.

La plupart des analyses d'erreur utilisent en fait une version simplifiée du premier modèle standard, selon laquelle $|\delta_1| < u$. Cependant, la borne $|\delta_1| \leqslant u/(1+u)$, bien que n'étant que très légèrement plus fine, permet de simplifier considérablement certaines démonstrations [55]. En base 2, le second modèle standard fournit une borne sur l'erreur absolue qui est à la fois atteignable et représentable : $|\widehat{r} - x \operatorname{op} y| \leqslant b$ avec $b = \widehat{r}\, 2^{-p} \in \mathbb{F}$. Plus généralement, ce second modèle permet de démontrer la validité de certaines analyses a posteriori (« *running error analysis* » [47, §3.3]) et, en combinaison avec le premier modèle, d'obtenir des bornes d'erreur très lisibles pour des calculs spécifiques mais importants comme la résolution d'un système linéaire triangulaire [47, §8.1].

Les modèles (5.11) et (5.12) ne reflètent bien évidemment pas toutes les propriétés contenues dans la règle d'arrondi correct (5.10) : ils ignorent notamment la structure des éléments de $\mathbb{F}$ décrite en (5.3) ainsi que des propriétés importantes des fonctions d'arrondi comme celles données en (5.7). Lors d'une analyse d'erreur a priori, l'idéal sera donc souvent d'utiliser ces propriétés « de bas niveau » en complément des modèles standard.

5.2.4 Quelques surprises typiques

Influence de la base. Notons tout d'abord que puisque les flottants sont des rationnels dont le dénominateur est une puissance entière de la base, une constante aussi simple que $1/10$ n'est pas représentable exactement en base 2 (et ce quelle que soit la précision p).

Un autre exemple typique est celui du calcul de la moyenne de deux nombres : si pour $x, y \in \mathbb{F}$ on évalue $(x+y)/2$ de façon usuelle, le résultat obtenu est $\widehat{r} = \circ(\circ(x+y)/2)$. En base 2, cette valeur approchée $\widehat{r}$ vérifie ce qu'on attend d'une moyenne :

$$\min\{x, y\} \leqslant \widehat{r} \leqslant \max\{x, y\}.$$

Cette double inégalité est une conséquence de (5.7) et du fait que $2x, 2y \in \mathbb{F}$ si $\beta = 2$. En revanche, elle n'est plus vraie en base 10 ; voir [10, 125] pour des contre-exemples et des façons de pallier cette difficulté.

Perte de certaines propriétés algébriques. La commutativité de $+$, $-$, $\times$ est préservée, comme conséquence directe de (5.10). En revanche, associativité et distributivité (de $\times$ par rapport à $\pm$) ne sont plus garanties. Par

exemple, si $x, y, z \in \mathbb{F}$ alors, en général, $\circ(\circ(x+y)+z) \neq \circ(x+\circ(y+z))$. Nous serons donc souvent amenés à considérer plusieurs schémas différents pour l'évaluation d'une même fonction. Même si, en principe, chaque schéma requiert une analyse d'erreur spécifique, nous verrons en section 5.3 que dans certains cas il est possible d'établir des bornes d'erreur valables indépendamment de l'ordre des opérations.

Élimination catastrophique. Même si chaque opération élémentaire bénéficie de l'arrondi correct et donc d'une erreur relative au plus u, en enchaîner deux peut suffire à produire un résultat dont l'erreur relative vaut 1 — ce qui signifie qu'aucun de ses chiffres ne peut être considéré comme correct. Ce phénomène, appelé *élimination catastrophique*, se produit par exemple pour $\widehat{r} = \mathrm{RN}(\mathrm{RN}(x+y)+z)$ avec $(x,y,z) = (1, u/\beta, -1) \in \mathbb{F}^3$. En effet, $\widehat{r} = 0$ et, la somme exacte r étant non nulle, l'erreur relative est alors $|\widehat{r}-r|/|r| = 1$. Cette perte quasi instantanée de toute qualité numérique est liée à la notion plus générale de *conditionnement* ; nous renvoyons à [15, 75] pour une présentation détaillée de ce concept.

Éviter ou retarder l'apparition de ce genre de phénomène peut requérir diverses modifications, de l'arithmétique ou du calcul lui-même : augmenter la précision p, récrire l'algorithme (comme remplacer $a^2 - b^2$ par $(a+b)(a-b)$, par exemple) ou encore introduire des techniques de *compensation* consistant à calculer certaines erreurs d'arrondi et à s'en servir pour améliorer la qualité du résultat initial [82]. Ce dernier aspect sera illustré en section 5.3.4 pour le calcul précis d'expressions de la forme $ab + cd$.

5.2.5 Opérations exactes et erreurs représentables

Nous terminons cette partie en présentant quelques situations où certaines quantités sont calculées de façon *exacte* en arithmétique à virgule flottante. Les propriétés ci-dessous peuvent s'avérer cruciales pour concevoir des algorithmes très précis et démontrer qu'ils le sont vraiment.

Opérations exactes. Si x et y sont dans $\mathbb{F}$, leur produit sera calculé exactement par (5.10) dès que le produit $M_x M_y$ de leurs mantisses entières est lui-même dans $\mathbb{F}$ (des cas particuliers étant le produit d'un flottant par une puissance entière de la base, et le produit d'entiers x et y tels que $|xy| \leqslant \beta^p$). L'analogue pour la soustraction est le théorème de Sterbenz [125, §4.3] :

Théorème 5.2.2. *Si $x, y \in \mathbb{F}$ sont tels que $y/2 \leqslant x \leqslant 2y$ alors $x - y \in \mathbb{F}$.*

Ce résultat, valable quelle que ce soit la base, nous dit que la différence de deux flottants suffisamment proches l'un de l'autre est elle-même un flot-

tant et sera donc calculée exactement. On peut le démontrer très simplement grâce aux propriétés de $\mathbb{F}$ vues en section 5.2.1 : quitte à échanger x et y, nous pouvons supposer que $0 \leqslant y \leqslant x \leqslant 2y$. Si $y = 0$ alors $x - y = x \in \mathbb{F}$. Sinon, $0 < \mathrm{ulp}(y) \leqslant \mathrm{ulp}(x)$ et donc, d'après (5.6), $x - y \in \mathrm{ulp}(y)\mathbb{Z}$. Ainsi, $x - y = M\beta^E$ avec M et E deux entiers tels que $\beta^E = \mathrm{ulp}(y)$ et, d'après (5.5), $|M| = (x-y)/\mathrm{ulp}(y) \leqslant y/\mathrm{ulp}(y) < \beta^p$.

Erreurs représentables et transformations exactes associées. Une autre conséquence remarquable de la règle d'arrondi correct (5.10) est l'implication suivante, selon laquelle l'erreur absolue commise en additionnant ou multipliant deux flottants est elle-même un flottant :

$$x, y \in \mathbb{F}, \quad \mathrm{op} \in \{+, \times\} \Rightarrow x \,\mathrm{op}\, y - \mathrm{RN}(x \,\mathrm{op}\, y) \in \mathbb{F}.$$

Ce résultat se démontre de la même manière que le théorème 5.2.2 et, dans le cas de la multiplication, reste valable pour les arrondis dirigés.

Étant donnés x, y et $\widehat{r} = \mathrm{RN}(x \,\mathrm{op}\, y)$, on sait de plus calculer l'erreur correspondante $e = x \,\mathrm{op}\, y - \widehat{r}$ à l'aide de seulement quelques opérations élémentaires en précision p (et donc sans recourir à une précision étendue). Pour la multiplication, il suffit d'utiliser l'opération FMA de la façon suivante :

$$\widehat{r} := \mathrm{RN}(xy); \qquad e := \mathrm{RN}(xy - \widehat{r}). \tag{5.13}$$

Pour l'addition, Kahan [58] et Dekker [29] ont remarqué que deux opérations supplémentaires suffisent dès que $\beta \in \{2, 3\}$ et $|x| \geqslant |y|$: [2]

$$\widehat{r} := \mathrm{RN}(x + y); \qquad \widehat{y} := \mathrm{RN}(\widehat{r} - x); \qquad e := \mathrm{RN}(y - \widehat{y}).$$

(Très schématiquement, l'idée est que si $\widehat{r}$ contient $x +$ (la partie haute de y), alors $\widehat{y}$ contient la partie haute de y et e contient la partie basse de y.) Chacun de ces algorithmes définit une *transformation exacte* ou *EFT* (pour « *error-free transform* » [104]), qui à $(x, y) \in \mathbb{F}^2$ associe $(\widehat{r}, e) \in \mathbb{F}^2$ tel que $x \,\mathrm{op}\, y = \widehat{r} + e$.

5.3 Analyse d'erreur a priori

5.3.1 Approche classique : l'analyse inverse de Wilkinson

La façon la plus courante d'analyser a priori l'effet des erreurs d'arrondi sur une solution approchée $\widehat{s}$ est l'*analyse inverse*, popularisée par Wilkinson dès les années soixante [136] et fondée sur la conséquence suivante,

2. Knuth [69] et Møller [89] ont montré que cinq additions supplémentaires (au lieu de deux) suffisent à s'affranchir de cette double condition.

très simple, du modèle standard (5.11) : pour $x, y \in \mathbb{F}$, l'approximation $\widehat{r} := \mathrm{RN}(x \operatorname{op} y)$ produite à chaque opération élémentaire est le *résultat exact de cette opération sur des données « proches »* $\widetilde{x}, \widetilde{y} \in \mathbb{R}$, c'est-à-dire

$$\widehat{r} = \widetilde{x} \operatorname{op} \widetilde{y}, \quad \widetilde{x} := x(1+\delta_x), \quad \widetilde{y} := y(1+\delta_y), \quad |\delta_x|, |\delta_y| \leqslant u. \tag{5.14}$$

Par exemple, on déduit de (5.11) que $\mathrm{RN}(x+y)$ est la somme exacte de $x(1+\delta_1)$ et $y(1+\delta_1)$. Dans le cas de la multiplication, on peut voir $\mathrm{RN}(xy)$ comme le produit exact de x et $y(1+\delta_1)$, ou de $x(1+\delta_1)$ et y, ou encore, si cela s'avère plus pratique, de $x(1+\delta_1)^{1/2}$ et $y(1+\delta_1)^{1/2}$.

Principe. Une analyse inverse consiste à appliquer (5.14) à chacune des opérations élémentaires utilisées pour produire $\widehat{s}$ et, plutôt que chercher à montrer que l'écart entre cette solution approchée $\widehat{s}$ et la solution exacte du problème est « petit » (ce qui peut s'avérer difficile, voire impossible), elle cherche à

— exprimer $\widehat{s}$ comme une solution exacte du même problème pour des données perturbées, et à
— majorer la taille, appelée *erreur inverse*, des plus petites perturbations possibles (pour une norme donnée).

Schématiquement, cette approche permet de considérer que commettre des erreurs d'arrondi équivaut à perturber les données du problème et, ainsi, de simplifier l'analyse numérique d'un algorithme en la ramenant à une analyse de la sensibilité du problème auquel l'algorithme s'applique. De plus, dans le cas de données inexactes (ce qui arrive souvent en pratique), si l'erreur inverse a le même ordre de grandeur que cette incertitude sur les données, on peut alors considérer que la solution calculée $\widehat{s}$ est tout à fait satisfaisante : elle pourrait être la solution exacte pour les données exactes.

Exemples. Étant donnés $x_1, \ldots, x_n \in \mathbb{F}$, considérons tout d'abord l'évaluation du produit $s = \prod_{i=1}^n x_i$ à l'aide de $n-1$ multiplications. On montre facilement par récurrence sur n que, pour tout ordre d'évaluation, il existe $n-1$ réels δ_i tels que la valeur calculée vérifie $\widehat{s} = s(1+\theta_{n-1})$ avec

$$1+\theta_{n-1} := \prod_{i=1}^{n-1}(1+\delta_i), \qquad |\delta_i| \leqslant u. \tag{5.15}$$

Dans ce cas, la situation est idéale au sens où à la fois l'*erreur directe* (c'est-à-dire l'écart relatif entre la solution approchée et la solution exacte) et l'erreur inverse sont, pour n fixé et $u \to 0$, toujours en $O(u)$. En effet, d'une part nous déduisons de (5.15) que quelles que soient les valeurs des x_i,

$$|\widehat{s} - s| \leqslant \alpha |s|, \qquad \alpha := (1+u)^{n-1} - 1 = (n-1)u + O(u^2); \tag{5.16}$$

d'autre part, $\widehat{s}$ se récrit aussi comme le produit exact de n réels $\widetilde{x}_i$, définis par exemple par $\widetilde{x}_1 = x_1$ et $\widetilde{x}_i = x_i(1+\delta_{i-1})$ pour $i \geqslant 2$, et donc obtenus par des perturbations relatives des x_i d'au plus u.

Considérons maintenant le cas, plus délicat, de l'évaluation de la somme $s = \sum_{i=1}^n x_i$. L'exemple d'élimination catastrophique donné en section 5.2.4 indique qu'on ne peut plus espérer majorer l'erreur directe comme en (5.16) pour tous les x_i. Cependant, nous allons voir que pour n fixé l'erreur inverse est encore en $O(u)$. Par exemple, si l'évaluation se fait « de gauche à droite » selon le schéma $(\cdots((x_1+x_2)+x_3)+\cdots)+x_n$, on montre facilement en appliquant $n-1$ fois (5.14) que la valeur obtenue s'écrit

$$\widehat{s} = (x_1+x_2)(1+\theta_{n-1}) + \sum_{i=3}^{n} x_i(1+\theta_{n-i+1}),$$

chaque θ_i étant une expression de la forme (5.15). Par conséquent, $\widehat{s}$ est la somme exacte des n réels $\widetilde{x}_i$ tels que $\widetilde{x}_1 = x_1(1+\theta_{n-1})$ et $\widetilde{x}_i = x_i(1+\theta_{n-i+1})$ pour $i \geqslant 2$, et on déduit que l'erreur inverse est majorée par $\max_i |\theta_i| \leqslant \alpha$ avec α défini en (5.16). En procédant par récurrence sur n, on obtient cette conclusion indépendamment de l'ordre d'évaluation de la somme.

L'analyse inverse que nous venons de faire pour la somme de n nombres flottants permet, par simple application de l'inégalité triangulaire, de majorer l'erreur directe de la façon suivante : si $s \neq 0$ alors

$$\frac{|\widehat{s}-s|}{|s|} \leqslant \alpha C, \qquad C := \frac{\sum_{i=1}^n |x_i|}{|\sum_{i=1}^n x_i|}. \tag{5.17}$$

Le nombre C, appelé *conditionnement*, dépend du problème (sommer n nombres) et de ses données (les x_i), mais ne dépend pas de l'algorithme utilisé (la façon de « parenthéser » la somme des x_i). Si $C = O(1)$ alors nous avons la garantie a priori que l'erreur directe sera petite (de l'ordre de u pour n fixé) et qu'ainsi $\widehat{s}$ sera une excellente approximation de la somme exacte. En revanche, si $C \gtrsim 1/u$ alors $\alpha C \gtrsim 1$, indiquant le risque que la valeur calculée $\widehat{s}$ soit totalement fausse ; c'est précisément ce qui se produit lors de l'élimination catastrophique en section 5.2.4, pour laquelle $\alpha \approx 2u$ et $C \approx 2\beta/u$.

Avantages et inconvénients. L'analyse inverse a été développée bien au-delà des deux exemples très simples que nous venons de voir et, comme en témoignent les 600 et quelques pages du livre de Higham [47], cette approche s'est révélée d'une extrême utilité pour prédire et expliquer le comportement de nombreux algorithmes, notamment en algèbre linéaire.

Sur le plan de l'arithmétique flottante, l'analyse inverse a l'avantage d'une certaine simplicité en ne faisant appel, la plupart du temps, qu'aux modèles standard (la complexité des analyses venant plutôt des techniques d'analyse matricielle, parfois sophistiquées, mises en œuvre pour réussir à rendre les bornes d'erreur suffisamment fines et lisibles).

Bien sûr, se limiter aux modèles (5.11) et (5.12) signifie aussi que l'on se prive des nombreuses propriétés « de bas niveau » qu'implique la règle d'arrondi correct (5.10). Dans les sous-sections qui suivent, nous allons voir que la prise en compte de (certaines de) ces propriétés permet au moins deux choses : d'une part, obtenir des bornes plus fines et plus concises que, par exemple, celle en (5.16) ; d'autre part, prouver que certains algorithmes sont toujours très précis.

5.3.2 Somme de n nombres flottants

Reprenons ici l'exemple de l'évaluation de $s = \sum_{i=1}^{n} x_i$ pour $x_i \in \mathbb{F}$. La borne d'erreur (5.17) est facile à obtenir, facile à interpréter et valable quel que soit l'ordre d'évaluation ; de plus, il existe des x_i pour lesquels le ratio erreur/(borne d'erreur) tend vers 1 lorsque $u \to 0$. (Prendre par exemple $x_1 = 1$ et $x_i = u$ pour $i \geqslant 2$.) Le seul défaut de cette borne concerne le terme α, dont la forme initiale $(1+u)^{n-1} - 1$ ne sera pas facile à manipuler lors d'analyses de plus haut niveau et dont la réécriture $(n-1)u + O(u^2)$ ne permet pas une majoration rigoureuse. (Notons aussi que remplacer u par $u/(1+u)$ ne suffit pas à supprimer les termes en $O(u^2)$ lorsque $n \geqslant 5$.)

Notation de Higham. Un compromis désormais classique consiste à utiliser la notation suivante, utilisée systématiquement par Higham dans [47] :

$$\gamma_k := \frac{ku}{1-ku} \quad \text{si} \quad ku < 1.$$

Ainsi, on pourra remplacer α par la borne γ_{n-1}, à la fois concise, rigoureuse et facile à propager grâce à des inégalités comme $\gamma_k + \gamma_\ell + \gamma_{k\ell} \leqslant \gamma_{k+\ell}$. Notons cependant que γ_{n-1} est encore de la forme $(n-1)u + O(u^2)$ et que la dimension n doit cette fois vérifier $(n-1)u < 1$.

Borne de Rump. Plus récemment, il a été observé par Rump [117] que pour la sommation « de gauche à droite », l'expression $\alpha = (1+u)^{n-1} - 1$ peut en fait toujours être remplacée par

$$\alpha = (n-1)u.$$

Cela conduit à une borne d'erreur plus fine et plus simple, sans terme quadratique en u et sans condition sur la valeur de n.

Pour établir ce résultat, Rump utilise non seulement le modèle standard (5.11), mais aussi la propriété suivante, qui découle directement de la définition d'arrondi « au plus proche » (5.8) appliquée à la somme de deux nombres flottants :

$$x, y \in \mathbb{F} \Rightarrow |\mathrm{RN}(x+y) - (x+y)| \leqslant \min\{|x|, |y|\}.$$

Plus précisément, cette propriété est exploitée dans une récurrence sur n visant à majorer l'erreur directe $|\widehat{s} - \sum_{i=1}^{n} x_i|$ par $(n-1)u\sum_{i=1}^{n}|x_i|$: on l'utilise si $|x_n| \leqslant u\sum_{i<n}|x_i|$, l'autre cas étant traité par le modèle standard.

Une généralisation de cette approche à tout ordre d'évaluation a été proposée dans [54, §3], permettant d'aboutir au résultat général suivant.

Théorème 5.3.1. *Soient $x_1, \ldots, x_n \in \mathbb{F}$. Si $\widehat{s} \in \mathbb{F}$ est le résultat de l'évaluation de $\sum_{i=1}^{n} x_i$ en arrondi « au plus proche » et pour un ordre quelconque, alors*

$$\left| \widehat{s} - \sum_{i=1}^{n} x_i \right| \leqslant (n-1)u \sum_{i=1}^{n} |x_i|.$$

5.3.3 Briques de base pour l'algèbre linéaire et les polynômes

Des améliorations similaires à celle présentée dans le théorème 5.3.1 ont été obtenues pour plusieurs autres problèmes de base, listés dans la table 5.1. Les algorithmes auxquels ces bornes d'erreur s'appliquent sont les schémas classiques (décrits notamment dans [47]), et la signification de α dépend du problème : par exemple, pour le produit de matrices, α est tel que $|\widehat{C} - AB| \leqslant \alpha|A||B|$ avec $A \in \mathbb{F}^{*\times n}$ et $B \in \mathbb{F}^{n\times *}$, $\widehat{C}$ désignant la matrice effectivement calculée, et l'inégalité et les valeurs absolues s'appliquant à chaque coefficient des matrices concernées ; pour la résolution de systèmes triangulaires et les factorisations LU et de Cholesky, nous considérons les majorations d'erreur inverse usuelles $|\Delta T| \leqslant \alpha|T|$ pour $(T + \Delta T)\widehat{x} = b$, $|\Delta A| \leqslant \alpha|\widehat{L}||\widehat{U}|$ pour $\widehat{L}\widehat{U} = A + \Delta A$, et $|\Delta A| \leqslant \alpha|\widehat{R}^T||\widehat{R}|$ pour $\widehat{R}^T\widehat{R} = A + \Delta A$. (Les matrices T, $\widehat{U}$, $\widehat{R}$ sont dans $\mathbb{F}^{n\times n}$ et $\widehat{L} \in \mathbb{F}^{m\times n}$ avec $m \geqslant n$.) Notons que les expressions de α données dans la table 5.1 n'ont aucun terme en $O(u^2)$, contrairement à celles obtenues avec l'approche classique présentée en section 5.3.1. Dans les deux cas ($\star$), le prix à payer pour ces bornes plus simples et plus fines est une condition (nécessaire) sur n détaillée dans [118].

TABLE 5.1 – *Améliorations récentes de quelques bornes classiques. Sauf indication contraire, l'ordre des opérations n'est pas spécifié et ($\star$) signifie « si $n \lesssim u^{-1/2}$ ».*

Problème	Valeur de α		Référence
prod. scalaire, prod. de matrices	nu		[54]
norme euclidienne $(\sum_{i=1}^n x_i^2)^{1/2}$	$(\frac{n}{2}+1)u$		[55]
$Tx = b, \quad A = LU$	nu		[83]
$A = R^T R$	$(n+1)u$		[83]
produit $\prod_{i=1}^n x_i$	$(n-1)u$	($\star$)	[118]
schéma de Horner pour $\sum_{i=0}^n a_i x^i$	$2nu$	($\star$)	[118]

5.3.4 Algorithmes précis pour évaluer $ab + cd$

Considérons maintenant l'évaluation de $ab + cd$ pour $a, b, c, d \in \mathbb{F}$. Cette opération, qui a de nombreuses applications (arithmétique complexe, discriminants, prédicats d'orientation robustes en géométrie algorithmique,...), ne fait pas partie des fonctions pour lesquelles la norme IEEE 754 requiert ou recommande l'arrondi correct. Elle est donc susceptible d'être implantée surtout en logiciel, à l'aide d'opérations élémentaires comme $+$, $\times$ ou FMA. Pour ce faire, il faut cependant être un peu soigneux : en effet, un schéma classique comme $\mathrm{RN}(\mathrm{RN}(ab) + \mathrm{RN}(cd))$ ou, si un FMA est disponible, $\mathrm{RN}(ab + \mathrm{RN}(cd))$ peut conduire à une « élimination catastrophique » et ainsi donner un résultat totalement faux.

Algorithme de Kahan. Pour éviter ce genre de désagrément, Kahan a proposé l'algorithme suivant (décrit dans [47, p. 60]) :

$$\widehat{w} := \mathrm{RN}(cd); \quad \widehat{f} := \mathrm{RN}(ab + \widehat{w}); \quad e := \mathrm{RN}(cd - \widehat{w}); \quad \widehat{r} := \mathrm{RN}(\widehat{f} + e).$$

Le FMA est utilisé une première fois pour approcher $ab + cd$ à l'aide de $\widehat{f}$. Sur les $2p$ chiffres du produit exact cd, les p chiffres les moins significatifs sont « perdus » par arrondi, de sorte que $\widehat{f}$ pourra fournir une très mauvaise approximation de $ab + cd$ lorsque $ab \approx -cd$. Le FMA est alors utilisé une seconde fois, pour implanter une EFT du produit cd comme en (5.13) et obtenir ainsi l'erreur $e = cd - \widehat{w}$ exactement ; cette erreur est enfin ajoutée à l'approximation initiale $\widehat{f}$ afin d'en améliorer la qualité.

L'algorithme de Kahan est toujours très précis au sens où l'erreur

relative sur sa sortie est en $O(u)$, et nous allons voir que ce résultat s'obtient en combinant le modèle standard (5.11) et des propriétés « de bas niveau ».

Tout d'abord, on déduit facilement de $e = cd - \widehat{w}$ et de (5.11) que

$$|\widehat{r} - r| \leqslant 2u|r| + u|e|, \qquad r = ab + cd.$$

Le modèle standard implique de plus que $|e| \leqslant u|cd|$. Cependant, utiliser cette majoration pour $|e|$ conduit à la borne d'erreur relative $2u + u^2|cd|/|r|$, qui pour certaines valeurs de a, b, c, d est supérieure à 1.

Pour conclure que l'algorithme de Kahan est *toujours* très précis, on peut compléter cette analyse classique par la prise en compte de propriétés plus fines, selon le schéma suivant : si $cd \in \mathbb{F}$ ou $ad + \widehat{w} \in \mathbb{F}$, alors on vérifie aisément en invoquant (5.7) que $\widehat{r}$ est l'arrondi correct de la valeur exacte : $\widehat{r} = \mathrm{RN}(r)$; sinon, on peut remplacer la majoration $|e| \leqslant u|cd|$ vue plus haut par $|e| \leqslant (\beta - 1)|r|$. (Ce second point n'est pas trivial, mais découle de propriétés comme (5.3) et (5.6) ; voir [53].) On déduit dans les deux cas que si $r \neq 0$ alors $|\widehat{r} - r|/|r| = O(u)$ pour β fixé.

Algorithme de Cornea, Harrison, Tang. Pour conclure, notons qu'il existe une variante de l'algorithme de Kahan [23, p. 273] permettant de garantir que la valeur renvoyée pour $ab + cd$ est la même que celle renvoyée pour $cd + ab$. (Cette propriété peut être utile pour fournir une implantation de la multiplication de deux nombres complexes qui préserve la commutativité.) Nous renvoyons à [52] pour une analyse fine de cette variante, mêlant modèle standard, analyse en ufp et utilisation du théorème 5.2.2.

5.4 Encadrer les erreurs avec des intervalles

Motivation. L'analyse directe de l'erreur flottante *a posteriori* peut être automatisée, avec plus ou moins d'efforts et plus ou moins de succès. C'est en tout cas l'optique qui a présidé à la définition de l'arithmétique par intervalles et surtout de l'analyse par intervalles dans les années 1950-1960[3]. Très rapidement, cette préoccupation s'est doublée du besoin de savoir prendre en compte d'autres erreurs : les incertitudes, de mesure ou autres, sur les données des calculs. L'arithmétique par intervalles incorpore de façon indifférenciée ces différentes sources d'erreur.

3. Les origines historiques de l'arithmétique par intervalles étant sujettes à discussion, contentons-nous de citer Sunaga pour son mémoire de master en japonais, intitulé "Geometry of Numerals", en 1956 à l'Université de Tokyo, et Moore pour le livre [91] paru en 1966 reprenant ses travaux de thèse. Pour un historique détaillé, on peut consulter http://www.cs.utep.edu/interval-comp/, rubrique *Early Papers, by Others*, ainsi que l'article *The origin of interval arithmetic*, de Rump, présenté à SCAN 2016.

Avant de voir comment utiliser l'arithmétique par intervalles pour analyser les erreurs d'arrondi des calculs, il est important de définir cette arithmétique avec une approche purement mathématique, afin d'assurer le bien-fondé des définitions [4].

Cette section commence par une définition des intervalles, puis décrit comment calculer avec des intervalles et se termine par la propriété fondamentale, dite propriété d'inclusion, qui justifie l'intérêt de l'arithmétique par intervalles en général, et en particulier pour les majorations des erreurs dues à l'arithmétique flottante.

5.4.1 Arithmétique par intervalles : présentation

Le principe de l'arithmétique par intervalles est de manipuler et de calculer non avec des nombres, mais avec des intervalles.

Définition 5.4.1. *Un intervalle est un sous-ensemble connexe fermé de* $\mathbb{R}$.

Par exemple, $\emptyset$, $[-1,3]$, $]-\infty,2]$, $[5,+\infty[$ et $\mathbb{R}$ sont des intervalles. En revanche $]-1,3]$, $]0,+\infty[$ ou $[1,2]\cup[3,4]$ ne sont pas des intervalles : les deux premiers ne sont pas fermés et le dernier n'est pas connexe.

La norme IEEE 754 pour l'arithmétique flottante définit les infinis $+\infty$ et $-\infty$ et permet de calculer avec ces valeurs. Cependant les analyses d'erreur d'arrondi ne sont plus valides en présence de valeurs infinies. Nous supposerons désormais que toutes les valeurs manipulées sont finies.

On notera en gras l'intervalle : $\mathbf{x} = \{x \in \mathbf{x}\}$ et on notera ses extrémités par $\underline{x}$, $\bar{x}$ avec $\underline{x} \in \mathbb{R}$, $\bar{x} \in \mathbb{R}$: $\mathbf{x} = [\underline{x}, \bar{x}] = \{x : \underline{x} \leqslant x \leqslant \bar{x}\}$.

Définition 5.4.2. *Une opération n-aire sur les réels* $\mathrm{op} : \mathbb{R}^n \to \mathbb{R}$, *s'étend en une opération n-aire sur les intervalles en utilisant la théorie des ensembles :*

$$\mathrm{op}(\mathbf{x}_1,\ldots,\mathbf{x}_n) = \mathrm{Hull}\{\mathrm{op}(x_1,\ldots,x_n) \mid x_i \in \mathbf{x}_i, 1 \leqslant i \leqslant n\},$$

où Hull désigne l'enveloppe convexe de l'ensemble. On ne considère que les opérandes $(x_i)_{1\leqslant i\leqslant n}$ *appartenant au domaine de définition de* op.

Cette définition théorique peut s'incarner dans des formules que l'on

4. On s'appuiera pour cette partie sur les ouvrages introductifs de Moore [92], Moore, Kearfott et Cloud [93] ainsi que celui de Tucker [131].

peut obtenir simplement, en utilisant la monotonie des opérateurs :

$$\begin{aligned}
\mathbf{x}+\mathbf{y} &= [\underline{x}+\underline{y}, \bar{x}+\bar{y}] \\
\mathbf{x}-\mathbf{y} &= [\underline{x}-\bar{y}, \bar{x}-\underline{y}] \\
\mathbf{x}\cdot\mathbf{y} &= [\min(\underline{x}\cdot\underline{y}, \underline{x}\cdot\bar{y}, \bar{x}\cdot\underline{y}, \bar{x}\cdot\bar{y}), \max(\underline{x}\cdot\underline{y}, \underline{x}\cdot\bar{y}, \bar{x}\cdot\underline{y}, \bar{x}\cdot\bar{y})] \\
\mathbf{x}/\mathbf{y} &= [\min(\underline{x}/\underline{y}, \underline{x}/\bar{y}, \bar{x}/\underline{y}, \bar{x}/\bar{y}), \max(\underline{x}/\underline{y}, \underline{x}/\bar{y}, \bar{x}/\underline{y}, \bar{x}/\bar{y})] \text{ si } 0 \notin \mathbf{y} \\
\sqrt{\mathbf{x}} &= \begin{cases} [\sqrt{\underline{x}}, \sqrt{\bar{x}}] & \text{si } \underline{x} \geqslant 0, \\ [0, \sqrt{\bar{x}}] & \text{si } \underline{x} \leqslant 0 \leqslant \bar{x}, \\ \emptyset & \text{si } \bar{x} < 0. \end{cases}
\end{aligned}$$

Pour $\cdot$ et $/$, on décompose $\mathbf{x}$ et $\mathbf{y}$ selon le signe de leurs éléments et on utilise les monotonies partielles sur $\mathbf{x} \cap \mathbb{R}^-$, $\mathbf{x} \cap \mathbb{R}^+$, ... Toujours en utilisant les monotonies des opérations et des fonctions, on peut déterminer des formules pour calculer le logarithme, le sinus ou toute autre fonction d'un intervalle, elles sont simplement plus longues à écrire quand la fonction n'est pas monotone.

La propriété qui justifie l'intérêt de ces calculs est la suivante.

Propriété 5.4.3 (Théorème fondamental de l'arithmétique par intervalles, ou propriété d'inclusion, ou « Tu ne mentiras point »)**.** *Soit c un calcul portant sur n nombres réels $x_1, \ldots, x_n$, qui est donné par une expression arithmétique complètement parenthésée, ou encore par un DAG (*directed acyclic graph*). Remplacer dans cette représentation de c les nombres $x_1, \ldots, x_n$ par des intervalles $\mathbf{x}_1, \ldots, \mathbf{x}_n$: $c(\mathbf{x}_1, \ldots, \mathbf{x}_n)$, puis évaluer les opérations comme indiqué précédemment, conduit à un résultat $\mathbf{y}$ qui vérifie*

$$\mathbf{y} \supset \{c(x_1, \ldots, x_n) \mid x_i \in \mathbf{x}_i, i = 1, \ldots, n\}.$$

En d'autres termes, le résultat d'un calcul mené en arithmétique par intervalles contient, inclut (d'où le nom de cette propriété) le résultat exact du calcul sur les réels. Ce résultat « ne ment pas » au sens où le résultat exact ne peut pas se trouver ailleurs.

5.4.2 Implantations et difficultés

Ces formules permettent d'implanter les différentes opérations et fonctions utiles, si on dispose d'une arithmétique exacte pour le calcul des extrémités. Sinon, afin de préserver la propriété d'inclusion, on calcule un résultat $\widehat{\mathbf{z}} = [\widehat{\underline{z}}, \widehat{\bar{z}}]$ qui contient le résultat $\mathbf{z} = [\underline{z}, \bar{z}]$, autrement dit il faut que $\widehat{\underline{z}} \leqslant \underline{z}$ et que $\bar{z} \leqslant \widehat{\bar{z}}$. C'est possible si on utilise l'arithmétique flottante,

grâce aux arrondis dirigés : par exemple, si $\underline{x}$, $\bar{x}$, $\underline{y}$ et $\bar{y}$ sont des nombres flottants, pour calculer $\mathbf{x} + \mathbf{y} = [\underline{x}, \bar{x}] + [\underline{y}, \bar{y}] = [\underline{x} + \underline{y}, \bar{x} + \bar{y}]$ on retourne le plus petit intervalle (au sens de l'inclusion) qui contient le résultat exact, il s'agit de $[\mathrm{RD}(\underline{x} + \underline{y}), \mathrm{RU}(\bar{x} + \bar{y})]$.

Il semble que la présence des arrondis dirigés RD et RU dans la norme IEEE 754 pour l'arithmétique flottante ait été imposée par Kahan, afin de faciliter l'implantation de l'arithmétique par intervalles. Cependant, pendant longtemps les langages de programmation de haut niveau ne donnaient pas accès aisément aux modes d'arrondi (et ne l'offrent toujours pas de façon normalisée). Il fallait avoir recours à des appels à des routines en assembleur, ce qui n'était ni intuitif ni portable. La situation s'est améliorée au fil des années et, par exemple, en langage C les appels à `fegetround()` et `fesetround()` fonctionnent sur la majorité des architectures avec le fichier `fenv.h`.

Une autre difficulté liée à l'utilisation des modes d'arrondis dirigés concerne l'efficacité des programmes, ou plutôt leur inefficacité, avec des temps d'exécution multipliés par des facteurs allant de 10 à 50 ou même 1000, en comparaison des mêmes programmes utilisant uniquement l'arithmétique flottante. Une première explication est que chaque opération sur des intervalles nécessitant plusieurs opérations flottantes, le calcul avec des intervalles est intrinsèquement plus lent que le calcul flottant. Par exemple, comme il faut 2 opérations flottantes pour l'addition de deux intervalles, cette dernière est deux fois plus lente que l'addition flottante. Il faut même 8 opérations flottantes pour la multiplication d'intervalles puisqu'il faut effectuer les quatre produits en arrondi vers $-\infty$ pour calculer la borne gauche et en arrondi vers $+\infty$ pour la borne droite, et donc 8 fois plus de temps que pour une multiplication flottante. Mais cela n'explique pas tout. Ce ralentissement est pour une grande part dû au fait que, sur les architectures classiques telles que IA32 et IA64 (plus connues sous le nom de x86), le mode d'arrondi est une information gérée par un drapeau d'état (*status flag*). Lorsque ce drapeau est modifié, toutes les instructions en cours d'exécution dans le pipeline sont arrêtées et recommencées avec le nouveau mode d'arrondi en vigueur. Si chaque opération en arithmétique par intervalles requiert au moins un changement de mode d'arrondi, plus aucun calcul ne peut être vectorisé, ce qui explique la perte d'efficacité d'un à deux ordres de grandeur. Nous avons décrit dans [110] d'autres problèmes liés aux changements de modes d'arrondi qui se posent lorsque les calculs sont parallèles, nous ne les aborderons pas ici. Ce qui justifie la perte du troisième ordre de grandeur est le remplacement des calculs flottants optimisés (pour l'utilisation des niveaux de cache etc.) — typiquement, des routines BLAS ou Lapack — par des calculs souvent moins optimisés.

Sur des architectures moins classiques, telles que les GPU ou la récente architecture *Knight Landing* d'Intel, ces problèmes ne devraient plus se poser puisque le mode d'arrondi est précisé dans le code de l'opération. Avec des implantations en CUDA pour GPU, on observe des accélérations, par rapport aux calculs sur CPU classiques, de 2 à 100 pour l'algorithme de Newton dans [7] et jusqu'à 1000 pour un calcul AXPY dans [19].

Difficultés d'utilisation : perte de propriétés algébriques. D'autres difficultés concernent l'écriture d'algorithmes utilisant l'arithmétique par intervalles et elles n'ont pas de solution simple et systématique. Il s'agit du fait que, comparée à l'arithmétique sur les nombres réels, l'arithmétique par intervalles « perd » certaines propriétés algébriques. Tout d'abord, la formule pour l'addition de deux intervalles $\mathbf{x}$ et $\mathbf{y}$:

$$\mathbf{x} + \mathbf{y} = [\underline{x}, \bar{x}] + [\underline{y}, \bar{y}] = [\underline{x} + \underline{y}, \bar{x} + \bar{y}]$$

permet de constater qu'en général un intervalle $\mathbf{x}$ n'a pas d'opposé : si $\underline{x} \neq \bar{x}$, il n'existe pas d'intervalle $\mathbf{y} = [\underline{y}, \bar{y}]$ tel que $\mathbf{x} + \mathbf{y} = [0, 0]$. En effet, si $\underline{x} \neq \bar{x}$, la largeur $w(\mathbf{x}) = \bar{x} - \underline{x}$ est strictement positive. Comme la largeur de $\mathbf{x} + \mathbf{y}$ est la somme des largeurs de $\mathbf{x}$ et $\mathbf{y}$: $w(\mathbf{x} + \mathbf{y}) = w([\underline{x} + \underline{y}, \bar{x} + \bar{y}]) = (\bar{x} - \underline{x}) + (\bar{y} - \underline{y}) = w(\mathbf{x}) + w(\mathbf{y})$, on a $w(\mathbf{x} + \mathbf{y}) > w(\mathbf{y}) \geqslant 0 = w([0, 0])$, on ne peut pas trouver $\mathbf{y}$ tel que $\mathbf{x} + \mathbf{y} = [0, 0]$. Toute addition ou soustraction ne fait qu'accroître la largeur des intervalles. Une conséquence est que l'on ne peut pas utiliser une formule mathématique qui repose sur une simplification de la forme « $x - x = 0$ » lorsque l'arithmétique utilisée est l'arithmétique par intervalles. Or les formules de Strassen pour la multiplication rapide de matrices utilisent cette réécriture, tout comme les calculs de différences finies pour approcher des dérivées.

Pour la multiplication également, un intervalle $\mathbf{x} = [\underline{x}, \bar{x}] \not\ni 0$ n'a en général pas d'inverse : il n'existe pas toujours d'intervalle $\mathbf{y} = [\underline{y}, \bar{y}]$ tel que $\mathbf{x} \cdot \mathbf{y} = [\min(\underline{x} \cdot \underline{y}, \underline{x} \cdot \bar{y}, \bar{x} \cdot \underline{y}, \bar{x} \cdot \bar{y}), \max(\underline{x} \cdot \underline{y}, \underline{x} \cdot \bar{y}, \bar{x} \cdot \underline{y}, \bar{x} \cdot \bar{y})] = [1, 1]$. Comme $\mathbf{x} \not\ni 0$, $\underline{x}\bar{x} > 0$. Supposons $\underline{x} > 0$, le cas où $\bar{x} < 0$ est similaire. On peut s'attendre à ce que l'inverse $\mathbf{y}$ vérifie aussi $\underline{y} > 0$. La formule pour le produit $\mathbf{x} \cdot \mathbf{y}$ se simplifie dans ce cas en $\mathbf{x} \cdot \mathbf{y} = [\underline{x} \cdot \underline{y}, \bar{x} \cdot \bar{y}]$. Pour que $\mathbf{x} \cdot \mathbf{y}$ soit égal à $[1, 1]$, il faut donc que $\underline{y} = 1/\underline{x}$ et que $\bar{y} = 1/\bar{x}$. On a alors $\mathbf{y} = [1/\underline{x}, 1/\bar{x}]$ avec $1/\underline{x} \leqslant 1/\bar{x}$, ce qui n'est possible que si $\underline{x} = \bar{x}$ et $\underline{y} = \bar{y}$. Dans ce cas, $\mathbf{x}$ ne contient qu'un seul nombre réel. Comme pour l'addition, si $w(\mathbf{x}) > 0$ alors $\mathbf{x}$ n'a pas d'inverse.

Une dernière propriété algébrique importante qui n'est pas vérifiée par l'arithmétique par intervalles est la distributivité de la multiplication sur l'addition : on a simplement $\mathbf{x} \cdot (\mathbf{y} + \mathbf{z}) \subset \mathbf{x} \cdot \mathbf{y} + \mathbf{x} \cdot \mathbf{z}$. Il suffit d'un exemple

pour s'en convaincre : $[1,2] \cdot ([-5,-3] + [4,7]) = [1,2] \cdot [-1,4] = [-2,8]$ et $[1,2] \cdot [-5,-3] + [1,2] \cdot [4,7] = [-10,-3] + [4,14] = [-6,11] \supset [-2,8]$.

L'explication de ces pertes de propriétés est connue sous le nom de *décorrélation des variables* ou en anglais *variable dependency*. Revenons à la définition d'une opération, par exemple $\mathbf{x} - \mathbf{y} = \{x - y \mid x \in \mathbf{x}, y \in \mathbf{y}\}$. Si $\mathbf{y} = \mathbf{x}$, on a donc $\mathbf{x} - \mathbf{x} = \{x - y \mid x \in \mathbf{x}, y \in \mathbf{x}\}$: cet ensemble contient 0, obtenu dans le cas où $x = y$, mais aussi d'autres éléments, obtenus quand $x \neq y$. Si on avait en tête $x - x = 0$, c'est-à-dire le fait que les deux occurrences de x sont identiques, cette information est perdue lors de l'écriture $\mathbf{x} - \mathbf{x}$: les deux occurrences sont décorrélées.

L'absence de ces propriétés algébriques signifie que l'écriture des formules, des algorithmes utilisant l'arithmétique par intervalles est délicate et demande d'utiliser $x \cdot (y + z)$ plutôt que $x \cdot y + x \cdot z$ ou d'éviter de réécrire 0 en $x - x$. Les règles à suivre sont en résumé les suivantes : il est préférable de n'introduire des intervalles dans les formules que le plus tard possible, après avoir réalisé toutes les manipulations algébriques souhaitées sur les expressions portant sur des nombres réels, et le plus parcimonieusement possible, en conservant le plus possible des variables réelles et non intervalles. Différentes variantes de l'arithmétique par intervalles, telles que l'arithmétique affine ou les modèles de Taylor, ont été développées pour pallier ces problèmes de décorrélation des variables. Elles sortent du cadre strict de l'analyse *a posteriori* et automatique des erreurs d'arrondi, puisqu'elles font appel à des reformulations des expressions évaluées et elles entraînent donc d'autres erreurs d'arrondi que celles des expressions originales. Elles seront brièvement évoquées en fin de ce chapitre.

5.4.3 Augmenter la précision de calcul

Analyse a posteriori. Dans cette section, on dispose d'une expression et d'une arithmétique flottante pour l'évaluer. L'approche choisie consiste à remplacer, dans cette expression, les données par des intervalles, à bornes flottantes, les contenant, et les opérations par des opérations sur ces intervalles, implantées en utilisant l'arithmétique flottante. Le résultat $\widehat{\mathbf{r}}$ de ce calcul par intervalles est un intervalle à bornes flottantes qui contient le résultat exact r et les erreurs d'arrondi commises lors des calculs : l'intervalle $\widehat{\mathbf{r}}$ permet d'encadrer les erreurs d'arrondi commises lors de l'évaluation de l'expression. Plus précisément la largeur $w(\widehat{\mathbf{r}})$ majore l'erreur absolue commise sur r et on s'intéresse donc à cette largeur. Comme les erreurs d'arrondi commises lors de l'évaluation des bornes, pour chaque opération intermédiaire, sont également prises en compte, $\widehat{\mathbf{r}}$ encadre, outre les erreurs d'arrondi commises lors du calcul de r, toutes ces erreurs d'arrondi sur les

bornes. La largeur de $\widehat{\mathbf{r}}$ est donc fréquemment pessimiste — on parle de *surencadrement*.

Tout d'abord, quelle est la largeur du plus petit (pour l'inclusion) intervalle $\widehat{\mathbf{t}}$ à bornes flottantes contenant un nombre réel $t \in \mathbb{R}$? On a vu que $\mathrm{RD}(t)$ et $\mathrm{RU}(t)$ définissent l'unique paire d'éléments consécutifs de $\mathbb{F}$ (ou égaux si $t \in \mathbb{F}$) telle que $t \in [\mathrm{RD}(t), \mathrm{RU}(t)]$, on peut donc utiliser (5.4) pour majorer la largeur de $\widehat{\mathbf{t}}$ par $w(\widehat{\mathbf{t}}) = \mathrm{RU}(t) - \mathrm{RD}(t) \leqslant 2u\beta^e \leqslant 2u|t|$. La largeur des intervalles qui sont utilisés comme données est donc au pire proportionnelle à u et à la valeur absolue des données réelles.

Si maintenant on calcule $\widehat{\mathbf{r}} = \widehat{\mathrm{op}}(\mathbf{x}_1, \ldots, \mathbf{x}_n)$, on a $\widehat{\mathbf{r}} = [\mathrm{RD}(r_1), \mathrm{RU}(r_2)]$, où r_1 et r_2 sont donnés par les formules de la section 5.4.1, donc on ajoute à la largeur du résultat exact $[r_1, r_2]$ une quantité que l'on peut majorer comme en section 5.2.2. Par exemple, si $\mathbf{x} \geqslant 0$ et $\mathbf{y} \geqslant 0$, c'est-à-dire que $\underline{x} \geqslant 0$ et $\underline{y} \geqslant 0$, et si $\mathbf{r} = \mathbf{x} + \mathbf{y}$ et $\widehat{\mathbf{r}} = \widehat{\mathbf{x} + \mathbf{y}} = [\mathrm{RD}(\underline{x} + \underline{y}), \mathrm{RU}(\bar{x} + \bar{y})]$, alors

$$\begin{aligned} w(\widehat{\mathbf{r}}) = \mathrm{RU}(\bar{x} + \bar{y}) - \mathrm{RD}(\underline{x} + \underline{y}) &\leqslant (1 + 2u)(\bar{x} + \bar{y}) - (1 - 2u)(\underline{x} + \underline{y}) \\ &\leqslant w(\mathbf{x} + \mathbf{y}) + 4u\frac{(\underline{x} + \underline{y}) + (\bar{x} + \bar{y})}{2}. \end{aligned}$$

On notera $\mathrm{mid}(\mathbf{r})$ le milieu de $\mathbf{r}$: $\mathrm{mid}(\mathbf{r}) = (\underline{r} + \bar{r})/2$. La largeur relative du résultat de l'addition, $w(\widehat{\mathbf{r}})/\mathrm{mid}(\mathbf{r})$, est augmentée de $4u$ comparée à la largeur relative du résultat exact. Il en va de même pour les autres opérations. On voit apparaître un analogue de l'équation (5.11) de la modélisation standard pour le calcul par intervalles, avec un accroissement de la largeur relative des intervalles calculés d'un facteur $4u$.

Plus généralement, si l'expression r et chacune de ses sous-expressions sont lipschitziennes en les variables, la surestimation du résultat calculé $w(\widehat{\mathbf{r}}) - w(\mathbf{r})$ est au pire proportionnelle à u : lorsque la précision de calcul p augmente, la largeur du résultat diminue. On trouve la preuve, par induction sur le DAG qui définit r, dans Neumaier [100, Theorem 2.1.5].

Quelques exemples. Illustrons ce résultat sur deux exemples, à l'aide de la bibliothèque MPFI [109] (qui calcule en base 2). Commençons par un exemple sans surprise : « vérifions numériquement » la formule de Machin :

$$\frac{\pi}{4} = 4\arctan(\frac{1}{5}) - \arctan(\frac{1}{239})$$

en calculant la différence entre $\pi/4$ et le terme de droite. On observe bien une largeur du résultat proportionnelle à l'unité d'erreur d'arrondi.

```
Enter the computing precision: 20
```

```
Difference = [-1.9073487e-6,1.9073487e-6]

Enter the computing precision: 40
Difference = [-1.8189894035459e-12,1.8189894035459e-12]

Enter the computing precision: 80
Difference = [-1.6543612251060553497428174e-24,1.6543612251060553497428174e-24]
```

Voici un autre exemple, la formule de Rump, construite pour provoquer des éliminations catastrophiques (vues en section 5.2.4) « en cascade », pour différentes valeurs d'exposants, tant que les calculs sont menés avec moins de 122 bits. On observe bien des encadrements très larges tant que la précision de calcul est insuffisante, puis très fins ensuite.

```
Rump's formula:
(333+3/4)*b^6 + a^2*(11*a^2*b^2-b^6-121*b^4-2) + 11/2*b^8 + a/2b
with a = 77617.0 and b = 33096.0
   single precision: 136652528711380592394066862325261926 4.000000
   double precision: 136652461123916627060868321645900595 2.000000

Enter the computing precision: 24
Eval. =  [-8.23972891e30,8.23972951e30]

Enter the computing precision: 53
Eval. =  [-5.9029581035870566e21,4.7223664828696463e21]

Enter the computing precision: 121
Eval. =  [-2.8273960599468213681411650954798162924,
           1.1726039400531786318588349045201837084]

Enter the computing precision: 122
Eval. =  [-8.2739605994682136814116509547981629201e-1,
          -8.2739605994682136814116509547981629162e-1]
```

Utiliser l'arithmétique par intervalles en faisant varier la précision de calcul ne demande pas une grande expertise, pas beaucoup de temps de développement et fournit des résultats intéressants, bien qu'inefficacement. C'est donc une approche assez automatique qui mérite d'être essayée.

Dans Gappa [5] [27, 12], l'arithmétique par intervalles en précision arbitraire est utilisée pour majorer les erreurs d'arrondi. Très schématiquement, elle intervient après une passe de réécriture des erreurs sous une forme facilement exploitable, en faisant apparaître explicitement les termes de la forme $x - \widehat{x}$ où x est la valeur exacte et $\widehat{x}$ la valeur approchée. La dernière phase de Gappa consiste à produire une preuve formelle, qui peut être vérifiée par l'assistant de preuve Coq, de la borne produite.

5. http://gappa.gforge.inria.fr/

5.4.4 Pour plus d'efficacité : représentation par centre et rayon

Représentation mid-rad. Dans les expériences numériques ci-dessus, on aimerait que la dernière valeur soit affichée comme

$$-0.827396059946821368141165095479816292 01 \pm 2.05 \cdot 10^{-37},$$

c'est-à-dire qu'elle soit représentée par son *milieu* (également appelé *centre*) et son *rayon*, ce qui s'appelle en anglais une représentation *mid-rad* pour *midpoint-radius*. Plusieurs bibliothèques utilisent cette représentation, telles que IntLab qui utilise deux flottants [114], ou ARB [56] et Mathemagix [78] qui stockent le centre avec une grande précision et le rayon avec une précision fixée.

Formules pour les opérations arithmétiques adaptées à la représentation mid-rad. Notons $\mathbf{x} = \langle x_m, x_r \rangle$ un intervalle $\mathbf{x}$ représenté par son milieu x_m et son rayon x_r (avec $x_r \geqslant 0$), $\mathbf{y} = \langle y_m, y_r \rangle$ et $\mathbf{z} = \langle z_m, z_r \rangle = \mathbf{x} \operatorname{op} \mathbf{y}$. On a par exemple

$$\mathbf{x} + \mathbf{y} = \langle x_m + y_m, x_r + y_r \rangle \qquad \mathbf{x} - \mathbf{y} = \langle x_m - y_m, x_r + y_r \rangle.$$

Pour implanter ces formules en arithmétique flottante, on calcule le milieu en arrondi au plus près. Le rayon est calculé en arrondi vers $+\infty$ afin de surestimer le rayon exact et de préserver la propriété d'inclusion ; de plus, il incorpore une majoration de l'erreur d'arrondi commise en calculant le milieu. Pour cela, ARB et Mathemagix ajoutent $u|z_m|$ au rayon.

Pour la multiplication, on trouve une formule exacte dans [100, Property 1.6.5]. Elle est assez compliquée et requiert au total 9 multiplications entre des réels. Très souvent, le rayon est surestimé à l'aide d'une formule plus simple. Dès 1999, Rump a proposé dans [113] de calculer z_m et z_r comme

$$z_m = x_m \cdot y_m \quad \text{et} \quad z_r = (|x_m| + x_r) \cdot y_r + x_r \cdot |y_m|.$$

Nguyen dans sa thèse [102] en 2011, Rump dans [115], Théveny dans sa thèse [128] en 2014, pour n'en citer que quelques-uns, proposent d'autres formules faisant intervenir de 2 à 7 multiplications entre des réels, et offrant des surestimations (en arithmétique exacte) plus ou moins larges.

Pour l'implantation flottante, les erreurs d'arrondi sont prises en compte dans le calcul du rayon : $z_m = \mathrm{RN}(x_m \cdot y_m)$ et $z_r = \mathrm{RU}((|x_m| + x_r) \cdot y_r + x_r \cdot |y_m| + u \cdot |z_m|)$.

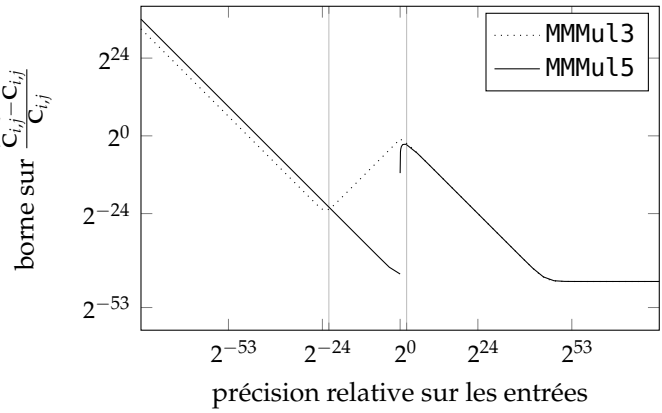


FIGURE 5.1 – *Comparaison des encadrements des largeurs relatives des rayons des matrices calculées par* `MMul3` *et* `MMul5`.

Efficacité des calculs. De telles formules sont particulièrement intéressantes lorsque l'on multiplie des matrices dont les coefficients sont des intervalles $\mathbf{A} \cdot \mathbf{B} = \langle A_m, A_r \rangle \cdot \langle B_m, B_r \rangle$; ces formules s'appliquent en remplaçant les nombres réels par des matrices et en appliquant la valeur absolue à chaque élément d'une matrice. On peut alors utiliser des routines optimisées et obtenir des produits efficaces pour les matrices à coefficients intervalles : le facteur de ralentissement dans [128] est de 3, bien loin des 1 à 3 ordres de grandeur mentionnés plus haut.

Cette approche est particulièrement intéressante pour les calculs parallèles. Si on stocke les rayons en simple précision plutôt qu'en double, chaque opération sur les rayons est très rapide et chaque mouvement de donnée est beaucoup moins gourmand, en temps et en énergie. Enfin, chaque valeur numérique, un centre et un rayon donc, donne lieu à plusieurs opérations arithmétiques avant d'être à nouveau stockée : l'intensité numérique du calcul est augmentée. Effectuer des calculs garantis pourrait devenir beaucoup moins pénalisant et donc plus attractif qu'à l'heure actuelle… mais il reste encore des efforts à faire pour optimiser les mouvements de données et l'intensité numérique.

Influence des erreurs d'arrondi. Le rayon du résultat est un encadrement des erreurs d'arrondi accumulées lors du calcul du centre et de son propre calcul. Dans les calculs en représentation mid-rad, quelle est la part relative de ces erreurs d'arrondi et de la largeur des intervalles en entrée ? Nous présentons ici quelques-uns des résultats de Théveny [128].

La figure 5.1 est tirée de [128, Fig. 4.14]. Le calcul considéré est un

produit de matrices $\mathbf{A} \cdot \mathbf{B}$ de dimensions 128×128 dont les coefficients sont des intervalles, choisies de sorte que l'incertitude relative sur les coefficients de $\mathbf{A}$, $\mathbf{A}_{i,j} = \langle A_{mi,j}, A_{ri,j} \rangle$ et de $\mathbf{B}$ soit majorée par e :

$$\frac{A_{ri,j}}{|A_{mi,j}|} \leqslant e, \quad \frac{B_{ri,j}}{|B_{mi,j}|} \leqslant e,$$

où la valeur e est atteinte. C'est cette valeur e qui figure en abscisse.

En ordonnée se trouve la même quantité pour $\mathbf{C}$ résultant du calcul $\mathbf{A} \cdot \mathbf{B}$ par deux algorithmes différents : il faut 3 produits de matrices pour `MMul3` et 5 pour `MMul5`. Avec une arithmétique exacte, une étude théorique a établi que `MMul5` est plus précis que `MMul3`. Expérimentalement, avec une arithmétique flottante double précision, on observe l'inverse quand les intervalles de $\mathbf{A}$ et $\mathbf{B}$ sont très fins : les rayons des éléments de $\mathbf{C}$ correspondent aux majorations des erreurs d'arrondi. Effectuant moins d'opérations, `MMul3` donne lieu à des erreurs d'arrondi plus faibles et donc à une courbe plus basse sur la partie gauche de la figure. Quand e croît, l'influence de la largeur des rayons des données prédomine et la plus faible surestimation du rayon, en arithmétique exacte, de `MMul5` devient manifeste. Les rayons cessent de correspondre aux erreurs d'arrondi des calculs.

Dans d'autres expériences, tirées de [128], les rayons des données sont fixés à un ulp des centres correspondants. Une grande majorité des rayons des résultats restent proches de l'ulp du coefficient correspondant : les erreurs d'arrondi initiales n'ont été que peu amplifiées par les calculs. Les autres sont proches de la borne théorique, obtenue *a priori*, et cela se produit pour les calculs dits *mal conditionnés* (évoqués en section 5.2.4), c'est-à-dire pour lesquels l'analyse prédit une large amplification des erreurs initiales.

5.4.5 Autres approches : arithmétique affine, modèles de Taylor

Terminons par quelques variantes de l'arithmétique par intervalles, qui réduisent le problème de décorrélation des variables. Ces variantes sortent du cadre strict de ce chapitre et sont mentionnées parce qu'elles sont employées en situation réelle ; voir la littérature citée ci-dessous et les séries de conférences SWIM et SCAN pour des exemples d'applications. Pour chaque opération, le temps d'exécution est de 2 à 3 ordres de grandeur supérieur au temps du calcul flottant, mais le gain en précision, et donc la réduction des calculs nécessaires à l'obtention d'un résultat garanti et précis, justifient ce surcoût.

Arithmétique affine. Cette arithmétique a été proposée par Comba et Stolfi en 1993 [20] et détaillée dans [126]. Le principe qui prévaut est de

remplacer chaque variable x (que ce soit une variable d'entrée ou un résultat de calcul intermédiaire) par une combinaison affine $x_0 + \sum_{i=1}^{n} x_i\varepsilon_i$, où les x_i sont des réels, et chaque ε_i est un symbole de bruit, indépendant des autres ε_j, $j \neq i$, et $\varepsilon_i \in [-1, 1]$. Voyons comment définir les opérations. Pour l'addition ou la soustraction de $x = x_0 + \sum_{i=1}^{n} x_i\varepsilon_i$ et $y = y_0 + \sum_{i=1}^{n} y_i\varepsilon_i$, on a :

$$z = x \pm y = x_0 \pm y_0 + \sum_{i=1}^{n} (x_i \pm y_i)\varepsilon_i : \begin{cases} z_0 & = x_0 \pm y_0, \\ z_i & = x_i \pm y_i, \quad i = 1, \ldots, n. \end{cases}$$

Conserver une expression symbolique permet par exemple de remédier au problème de l'expression $x - x$ qui s'évalue ainsi en 0.

Cela se complique pour la multiplication de deux variables : des termes de la forme $\varepsilon_i\varepsilon_j$ apparaissent, ils sont transformés en un nouveau terme $z_{n+1}\varepsilon_{n+1}$ qui nécessite la création d'une variable ε_{n+1}. Le même problème se pose pour la division et toute fonction non linéaire : $\sqrt{}$, exp, tan,... De nombreuses variantes existent pour ces opérations, qui visent à limiter le nombre de variables créées lors d'un calcul et aussi à incorporer les erreurs d'arrondi commises sur chaque coefficient z_i. Citons les implantations en MATLAB [116] ou en C++ dans IBEX[6].

Une application phare de l'arithmétique affine et d'autres variantes plus avancées (utilisant des zonotopes) est le logiciel Fluctuat, décrit par Putot dans [107] et Martel dans [84]. En quelques mots,

> *Fluctuat is a static analyzer by abstract interpretation, relying on the general-purpose zonotopic abstract domains, and dedicated to the analysis of numerical properties of programs.*

L'arithmétique affine permet d'encadrer les valeurs des résultats, les erreurs d'arrondi et ainsi de garder une trace des opérations où elles se sont produites, afin d'identifier à quel moment elles posent problème.

Modèles polynomiaux, modèles de Taylor. Un tel modèle utilise des développements polynomiaux (et non affines) en les variables d'entrée, avec une seule variable pour chaque variable d'entrée et en fixant à l'avance le degré maximal des polynômes utilisés. Les implantations accessibles sont peu nombreuses : l'implantation la plus ancienne est due à Eckmann et coauteurs [31], citons également COSY [108]. Enfin, on trouve des variations dans le choix des bases pour les approximations polynomiales : Bernstein chez Nataraj et Kotecha [98] pour l'obtention simple de bornes, Chebychev chez Joldes [57, Chap. 4] pour la qualité de l'approximation polynomiale.

6. http://www.ibex-lib.org/

5.5 Notes bibliographiques

En ce qui concerne l'arithmétique à virgule flottante, nous renvoyons à [95] pour une présentation plus complète, notamment des détails de la norme IEEE 754. Des exemples d'implantation de cette norme figurent dans les thèses de Brunie [14], Lauter [77] et Revy [111], pour différents contextes.

Concernant l'analyse inverse, une référence reste le livre de Higham [47] ; le lien entre erreur inverse et erreur directe via la notion de conditionnement est présenté dans [47, §1] et [75].

Dans les sections 5.2 et 5.3, l'accent a été mis sur l'arrondi « au plus proche ». Il existe cependant quelques résultats récents visant à étendre ces résultats aux arrondis dirigés [11, 105]. De même, nous avons supposé que tous les calculs se font en l'absence de dépassement de capacité, c'est-a-dire qu'ils ne génèrent ni *underflow*, ni *overflow*. La prise en compte de ces exceptions, souvent délicate, a néanmoins fait l'objet de travaux récents, comme par exemple [10]. Dans tous les cas, la vérification formelle de l'ensemble de ces analyses, telle que présentée dans [12], est toujours d'actualité.

En ce qui concerne l'arithmétique par intervalles, une lecture complémentaire qui s'impose est celle de la récente norme IEEE 1788-2015 [49]. Les implantations de cette norme sont en cours, avec la bibliothèque `libieee1788` [99] en C++, qui repose sur une arithmétique flottante à précision arbitraire, ou la bibliothèque en Octave [45].

Il existe aussi une branche entière de l'analyse numérique dévolue au calcul avec des intervalles, qui permet d'obtenir des résultats inaccessibles lorsque l'on calcule avec des réels. Citons notamment la méthode de Newton pour encadrer tous les zéros d'une fonction f sur un intervalle donné $\mathbf{x}_0$, et ce souvent avec une preuve de leur existence et de leur unicité, ou a contrario qui fournit une preuve de l'absence de zéros. L'arithmétique par intervalles permet également, grâce à son caractère ensembliste, de développer des algorithmes d'optimisation globale, on trouvera une introduction dans [43]. Enfin, elle est utilisée avec succès pour l'intégration garantie d'équations différentielles ordinaires, par exemple pour prouver que le système de Lorenz admet un attracteur [130]. Les variantes (arithmétique affine, modèles polynomiaux) de l'arithmétique par intervalles sont employées dans ce cas, comme par exemple l'arithmétique affine dans [2].

Remerciements. Ce travail a été financé en partie par l'Agence Nationale de la Recherche (projet ANR-13-INSE-0007 *MetaLibm*).

Bibliographie

[1] M. Albert, R. J. Nowakowski, and D. Wolfe. *Lessons in Play: an Introduction to Combinatorial Game Theory*. A K Peters, 2007.

[2] J. Alexandre dit Sandretto and A. Chapoutot. Validated explicit and implicit Runge-Kutta methods. *Reliable Computing*, 22:78–103, 2016.

[3] S. Amoroso and Y. N. Patt. Decision procedures for surjectivity and injectivity of parallel maps for tessellation structures. *Journal of Computer and System Sciences*, 6(5):448–464, 1972.

[4] K. R. Apt and E. Grädel. *Lectures in game theory for computer scientists*. Cambridge University Press, 2011.

[5] S. Arora and B. Barak. *Computational Complexity: a modern Approach*. Cambridge University Press, 2009.

[6] H. Barendregt. *The Lambda Calculus: its Syntax and Semantics*, volume 103 of *Studies in Logic and The Foundations of Mathematics*. North-Holland, 1984.

[7] P.-D. Beck and M. Nehmeier. Parallel interval Newton method on CUDA. In *International Workshop on Applied Parallel Computing*, pages 454–464. Springer, 2012.

[8] R. Berger. The undecidability of the domino problem. *Memoirs of the American Mathematical Society*, 66, 1966.

[9] E. R. Berlekamp, J. H. Conway, and R. K. Guy. *Winning ways for your mathematical plays*. A K Peters, 2001.

[10] S. Boldo. Formal verification of programs computing the floating-point average. In M. Butler, S. Conchon, and F. Zaïdi, editors, *Proc. 17th International Conference on Formal Engineering Methods*, volume 9407 of *Lecture Notes in Computer Science*, pages 17–32, Paris, France, 2015. Springer International Publishing.

[11] S. Boldo, S. Graillat, and J.-M. Muller. On the robustness of the 2Sum and Fast2Sum algorithms. Technical Report ensl-01310023, Hal, 2016.

[12] S. Boldo and G. Melquiond. Arithmétique des ordinateurs et preuves formelles. In P. Langlois, editor, *Informatique Mathématique : une photographie en 2013*, pages 189–220. Presses Universitaires de Perpignan, 2013.

[13] C. L. Bouton. Nim, a game with a complete mathematical theory. *Annals of Mathematics*, 3:35–39, 1905.

[14] N. Brunie. *Contributions to Computer Arithmetic and Applications to Embedded Systems*. Thèse de doctorat, École normale supérieure de Lyon, Lyon, France, mai 2014.

[15] P. Bürgisser and F. Cucker. *Condition*. Springer-Verlag Berlin Heidelberg, 2013.

[16] T. Ceccherini-Silberstein and M. Coornaert. *Cellular Automata and Groups*. Springer, 2010.

[17] B. Chopard and M. Droz. *Cellular automata modeling of physical systems*. Cambridge University Press, 1998.

[18] A. Church. *The calculi of lambda-conversion*, volume 6 of *Annals of Mathematical Studies*. Princeton, 1941.

[19] S. Collange, M. Daumas, and D. Defour. *GPU Computing Gems Jade Edition*, chapter Interval arithmetic in CUDA, pages 99–107. Morgan Kaufmann, 2011.

[20] J. L. D. Comba and J. Stolfi. Affine arithmetic and its applications to computer graphics. In *Proceedings of SIBGRAPI'93 - VI Simpósio Brasileiro de Computação Gráfica e Processamento de Imagens*, pages 9–18, 1993.

[21] J. H. Conway. *On numbers and games*. Academic Press, 1976.

[22] R. Cori and D. Lascar. *Logique mathématique, tome 1 : Calcul propositionnel; algèbre de Boole; calcul des prédicats*. Dunod, Paris, 2003.

[23] M. Cornea, J. Harrison, and P. T. P. Tang. *Scientific Computing on Itanium®-based Systems*. Intel Press, Hillsboro, OR, USA, 2002.

[24] K. Culik, II. An aperiodic set of 13 Wang tiles. *Discrete Mathematics*, 160(1-3):245–251, 1996.

[25] K. Culik, II, J. Pachl, and S. Yu. On the limit sets of cellular automata. *SIAM Journal on Computing*, 18(4):831–842, Aug. 1989.

[26] E. Czeizler and J. Kari. A tight linear bound on the neighborhood of inverse cellular automata. In *Automata, Languages and Programming, 32nd International Colloquium, ICALP 2005, Lisbon, Portugal, July 11-15, 2005.*, number 3580 in Lecture Notes in Computer Science, pages 410–420, 2005.

[27] M. Daumas and G. Melquiond. Certification of bounds on expressions involving rounded operators. *ACM Transactions on Mathematical Software*, 37(1):1–20, 2010.

[28] R. David, K. Nour, and C. Raffalli. *Introduction à la logique: théorie de la démonstration (2e édition)*. Dunod, 2004.

[29] T. J. Dekker. A floating-point technique for extending the available precision. *Numer. Math.*, 18:224–242, 1971.

[30] B. Durand. The surjectivity problem for 2D cellular automata. *Journal of Computer and System Sciences*, 49(3):718–725, 1994.

[31] J.-P. Eckmann, A. Malaspinas, and S. O. Kamphorst. *A Software Tool for Analysis in Function Spaces*, pages 147–167. Springer, 1991.

[32] J. Edmonds. Paths, trees, and flowers. *Canadian Journal of Mathematics*, 17:449–467, 1965.

[33] S. Even and R. E. Tarjan. A combinatorial problem which is complete in polynomial space. *Journal of the ACM*, 23(4), 1976.

[34] O. Finkel. Borel ranks and Wadge degrees of context free ω-languages. *Mathematical Structures in Computer Science*, 16(5):813–840, 2006.

[35] A. S. Fraenkel and D. Lichtenstein. Computing a perfect strategy for $n \times n$ chess requires time exponential in n. *Journal of Combinatorial Theory, Series A*, 31(2):199–214, 1981.

[36] A. S. Fraenkel, E. R. Scheinerman, and D. Ullman. Undirected edge geography. *Theoretical Computer Science*, 112(2):371–381, 1993.

[37] M. Gardner. Mathematical games: Cram, crosscram and quadraphage: new games having elusive winning strategies. *Scientific American*, 230, 1974.

[38] E. Grädel, W. Thomas, and T. Wilke, editors. *Automata, Logics, and Infinite Games. A Guide to Current Research*, volume 2500 of *Lecture Notes in Computer Science*. Springer-Verlag, 2003.

[39] T. Griffin. A formulae-as-types notion of control. In *POPL '90. Proceedings of the 17th ACM SIGPLAN-SIGACT symposium on Principles of programming languages*, pages 47–58. ACM Press, 1990.

[40] M. Guillermo and A. Miquel. Specifying Peirce's law. *Mathematical Structures in Computer Science*, 26(7):1269–1303, 2016.

[41] Y. Gurevich and I. Koryakov. Remarks on Berger's paper on the domino problem. *Siberian Mathematical Journal*, 13(2):459–463, 1972.

[42] R. K. Guy. Unsolved problems in combinatorial games. In R. J. Nowakowski, editor, *Games of No Chance*, volume 29 of *MSRI Publications*, pages 475–491. Cambridge University Press, 1996.

[43] E. R. Hansen and G. W. Walster. *Global optimization using interval analysis (2nd ed.)*. Marcel Dekker, 2003.

[44] G. A. Hedlund. Endomorphisms and Automorphisms of the Shift Dynamical Systems. *Mathematical Systems Theory*, 3(4):320–375, 1969.

[45] O. Heimlich. Interval arithmetic in GNU Octave. In *SWIM 2016: Summer Workshop on Interval Methods*, 2016.

[46] H. Herrlich. *Axiom of choice*. Springer, 2006.

[47] N. J. Higham. *Accuracy and Stability of Numerical Algorithms*. Society for Industrial and Applied Mathematics, Philadelphia, PA, USA, second edition, 2002.

[48] IEEE Computer Society. *IEEE Standard for Floating-Point Arithmetic, IEEE Standard 754-2008*. IEEE Computer Society, New York, Aug. 2008.

[49] IEEE Computer Society. *1788-2015 - IEEE Standard for Interval Arithmetic*. IEEE Computer Society, New York, June 2015.

[50] E. Jeandel. The periodic domino problem revisited. *Theoretical Computer Science*, 411(44):4010–4016, 2010.

[51] E. Jeandel and M. Rao. An aperiodic set of 11 Wang tiles. Technical Report 1506.06492, arXiv, 2015.

[52] C.-P. Jeannerod. A radix-independent error analysis of the Cornea-Harrison-Tang method. *ACM Trans. Math. Software*, 42(3):19:1–19:20, 2016.

[53] C.-P. Jeannerod, N. Louvet, and J.-M. Muller. Further analysis of Kahan's algorithm for the accurate computation of 2×2 determinants. *Mathematics of Computation*, 82(284):2245–2264, 2013.

[54] C.-P. Jeannerod and S. M. Rump. Improved error bounds for inner products in floating-point arithmetic. *SIAM Journal on Matrix Analysis and Applications*, 34(2):338–344, 2013.

[55] C.-P. Jeannerod and S. M. Rump. On relative errors of floating-point operations: optimal bounds and applications. *Mathematics of Computation*, To appear.

[56] F. Johansson. Arb: a C library for ball arithmetic. *ACM Communications in Computer Algebra*, 47(4):166–169, 2013.

[57] M. Joldes. *Rigorous Polynomial Approximations and Applications*. Thèse de doctorat, École Normale Supérieure de Lyon, 2011.

[58] W. Kahan. Further remarks on reducing truncation errors. *Comm. ACM*, 8(1):40, 1965.

[59] A. Kahr, E. F. Moore, and H. Wang. Entscheidungsproblem reduced to the $\forall\exists\forall$ case. *Proceedings of the National Academy of Sciences of the United States of America*, 48(3):365–377, 1962.

[60] J. Kari. The nilpotency problem of one-dimensional cellular automata. *SIAM Journal on Computing*, 21:571–586, 1992.

[61] J. Kari. Reversibility and surjectivity problems of cellular automata. *Journal of Computer and System Sciences*, 48(1):149–182, 1994.

[62] J. Kari. A small aperiodic set of wang tiles. *Discrete Mathematics*, 160(1-3):259–264, 1996.

[63] J. Kari. Infinite snake tiling problems. In *Developments in Language Theory, 6th International Conference, DLT 2002, Kyoto, Japan, September 18-21, 2002, Revised Papers*, number 2450 in Lecture Notes in Computer Science, pages 67–77, 2002.

[64] J. Kari. Theory of cellular automata: A survey. *Theoretical Computer Science*, 334(1):3–33, 2005.

[65] J. Kari. Snakes and cellular automata: Reductions and inseparability results. In *Computer Science - Theory and Applications. 6th International Computer Science Symposium in Russia, CSR 2011, St. Petersburg, Russia, June 14-18, 2011.*, number 6651 in Lecture Notes in Computer Science, pages 223–232. Springer, 2011.

[66] J. Kari and N. Ollinger. Periodicity and immortality in reversible computing. In *Mathematical Foundations of Computer Science 2008, 33rd International Symposium, MFCS 2008, Torun, Poland, August 25-29, 2008, Proceedings*, number 5162 in Lecture Notes in Computer Science, pages 419–430. Springer, 2008.

[67] A. Kechris. *Classical descriptive set theory*, volume 156. Springer-Verlag, 2012.

[68] S. C. Kleene. On the interpretation of intuitionistic number theory. *Journal of Symbolic Logic*, 10:109–124, 1945.

[69] D. E. Knuth. *The Art of Computer Programming, Volume 2, Seminumerical Algorithms*. Addison-Wesley, Reading, MA, USA, third edition, 1998.

[70] J. L. Krivine. *Lambda-calcul, types et modèles*. Masson, 1991.

[71] J.-L. Krivine. Realizability in classical logic. In *Interactive models of computation and program behaviour*, volume 27 of *Panoramas et synthèses*, pages 197–229. Société Mathématique de France, 2009.

[72] J.-L. Krivine. Realizability algebras: a program to well order R. *Logical Methods in Computer Science*, 7:1–47, 2011.

[73] J.-L. Krivine. Realizability algebras II: new models of ZF + DC. *Logical Methods for Computer Science*, 8(1):1–28, 2012.

[74] P. Kůrka. *Topological and symbolic dynamics*. Société Mathématique de France, 2003.

[75] P. Langlois. *Outils d'analyse numérique pour l'automatique – Chapitre 1*, pages 19–52. Traité IC2 dirigé par Alain Barraud. Hermès Science, 2002.

[76] U. Larsson, R. J. Nowakowski, and C. Santos. When waiting moves you in scoring combinatorial games. Technical Report 1505.01907, arXiv, 2015.

[77] C. Q. Lauter. *Arrondi correct de fonctions mathématiques : fonctions univariées et bivariées, certification et automatisationArrondi correct de fonctions mathématiques : fonctions univariées et bivariées, certification et automatisation*. Thèse de doctorat, École normale supérieure de Lyon, Lyon, France, octobre 2008.

[78] G. Lecerf and J. van der Hoeven. Evaluating Straight-Line Programs over Balls. In *23rd IEEE Symposium on Computer Arithmetic*, pages 142–149, 2016.

[79] D. Lichtenstein and M. Sipser. Go is polynomial-space hard. *Journal of the ACM*, 27:393–401, 1980.

[80] D. Lind. *Symbolic Dynamics and its Applications*, volume 60 of *Proceedings of Symposia in Applied Mathematics*, chapter Multi-dimensional symbolic dynamics, pages 61–80. American Mathematical Society, 2004.

[81] D. A. Lind and B. Marcus. *An Introduction to Symbolic Dynamics and Coding*. Cambridge University Press, 1995.

[82] N. Louvet. *Algorithmes compensés en arithmétique flottante : précision, validation, performances*. Thèse de doctorat, Université de Perpignan, Perpignan, France, Nov. 2007.

[83] S. M. Rump and C.-P. Jeannerod. Improved backward error bounds for LU and Cholesky factorizations. *SIAM J. Matrix Anal. Appl.*, 35(2):684–698, 2014.

[84] M. Martel. Interprétation abstraite pour la précision numérique. In P. Langlois, editor, *Informatique Mathématique : une photographie en 2013*, pages 145–185. Presses Universitaires de Perpignan, 2013.

[85] D. A. Martin. Borel determinacy. *The Annals of Mathematics*, 102(2):363–371, 1975.

[86] A. Miquel. Classical program extraction in the calculus of constructions. In *Computer Science Logic*, pages 313–327. Springer, 2007.

[87] A. Miquel. Existential witness extraction in classical realizability and via a negative translation. *Logical Methods for Computer Science*, 7(2):1–52, 2010.

[88] A. Miquel. Forcing as a program transformation. In *Symposium on Logic in Computer Science (LICS)*, pages 197–206. IEEE Computer Society, 2011.

[89] O. Møller. Quasi double-precision in floating point addition. *BIT*, 5:37–50, 1965.

[90] E. F. Moore. Machine Models of Self-Reproduction. In *Proceedings of Symposia in Applied Mathematics*, volume 14, pages 17–33. American Mathematical Society, 1962.

[91] R. Moore. *Interval analysis*. Prentice Hall, 1966.

[92] R. Moore. *Methods and applications of interval analysis*. SIAM Studies in Applied Mathematics, 1979.

[93] R. Moore, R. Kearfott, and M. Cloud. *Introduction to Interval Analysis*. SIAM, 2009.

[94] M. Morse and G. Hedlund. *Symbolic Dynamics*. Johns Hopkins University Press, 1938.

[95] J.-M. Muller, N. Brisebarre, F. de Dinechin, C.-P. Jeannerod, V. Lefèvre, G. Melquiond, N. Revol, D. Stehlé, and S. Torres. *Handbook of Floating-Point Arithmetic*. Birkhäuser Boston, 2010.

[96] F. Murlak. The Wadge hierarchy of deterministic tree languages. *Logical Methods in Computer Science*, 4(4):1–44, 2008.

[97] J. Myhill. The converse of Moore's garden-of-Eden theorem. In *Proceedings of the American Mathematical Society*, volume 14, pages 658–686. American Mathematical Society, 1963.

[98] P. S. V. Nataraj and K. Kotecha. Higher order convergence for multi-dimensional functions with a new Taylor-Bernstein form as inclusion function. *Reliable Computing*, 9:185–203, 2003.

[99] M. Nehmeier, J. Wolff von Gudenberg, and J. Pryce. On Implementing the IEEE Interval Standard P1788. In *FEMTEC 2013: 4th International Congress on Computational Engineering and Sciences*, page 103, 2013.

[100] A. Neumaier. *Interval methods for systems of equations*. Cambridge University Press, 1990.

[101] J. V. Neumann. *Theory of Self-Reproducing Automata*. University of Illinois Press, Urbana, Illinois, 1966.

[102] H. D. Nguyen. *Efficient algorithms for verified scientific computing : Numerical linear algebra using interval arithmetic*. Thèse de doctorat, Ecole normale supérieure de Lyon, Jan. 2011.

[103] D. Niwiński and I. Walukiewicz. A gap property of deterministic tree languages. *Theoretical Computer Science*, 303(1):215–231, 2003.

[104] T. Ogita, S. M. Rump, and S. Oishi. Accurate sum and dot product. *SIAM J. Sci. Comput.*, 26(6):1955–1988, 2005.

[105] K. Ozaki, T. Ogita, F. Bünger, and S. Oishi. Accelerating interval matrix multiplication by mixed precision arithmetic. *Nonlinear Theory and Its Applications*, 6(3):364–376, 2015.

[106] D. Perrin and J.-E. Pin. *Infinite words. Automata, semigroups, logic and games*, volume 141 of *Pure and applied Mathematics*. Academic Press, 2004.

[107] S. Putot. *Static Analysis of Numerical Programs and Systems*. Habilitation à diriger des recherches, Université de Paris-Sud, 2012.

[108] N. Revol, K. Makino, and M. Berz. Taylor models and floating-point arithmetic: proof that arithmetic operations are validated in COSY. *Journal of Logic and Algebraic Programming*, 64:135–154, 2005.

[109] N. Revol and F. Rouillier. Motivations for an arbitrary precision interval arithmetic and the MPFI library. *Reliable Computing*, 11(4):275–290, 2005.

[110] N. R. Revol and P. Théveny. Numerical reproducibility and parallel computations: Issues for interval algorithms. *IEEE Transactions on Computers*, 63(8):1915–1924, 2014.

[111] G. Revy. *Implementation of binary floating-point arithmetic on embedded integer processors – Polynomial evaluation-based algorithms and certified code generation*. Thèse de doctorat, École normale supérieure de Lyon, Lyon, France, décembre 2009.

[112] R. M. Robinson. Undecidability and nonperiodicity for tilings of the plane. *Inventiones Mathematicae*, 12:177–209, 1971.

[113] S. Rump. Fast and parallel interval arithmetic. *BIT*, 39(3):534–554, 1999.

[114] S. Rump. INTLAB - INTerval LABoratory. In T. Csendes, editor, *Developments in Reliable Computing*, pages 77–104. Kluwer Academic Publishers, Dordrecht, 1999.

[115] S. Rump. Fast interval matrix multiplication. *Numerical Algorithms*, 1(61):1–34, 2012.

[116] S. Rump and M. Kashiwagi. Implementation and improvements of affine arithmetic. *Nonlinear Theory and Its Applications*, 6(3):341–359, 2015.

[117] S. M. Rump. Error estimation of floating-point summation and dot product. *BIT*, 52(1):201–220, 2012.

[118] S. M. Rump, F. Bünger, and C.-P. Jeannerod. Improved error bounds for floating-point products and Horner's scheme. *BIT*, 56(1):293–307, 2016.

[119] S. M. Rump, T. Ogita, and S. Oishi. Accurate floating-point summation, Part I: Faithful rounding. *SIAM J. Sci. Comput.*, 31(1):189–224, 2008.

[120] T. J. Schaeffer. On the complexity of some two-person perfect-information games. *Journal of Computer and System Sciences*, 16(2):185–225, 1978.

[121] V. Selivanov. Fine hierarchy of regular ω-languages. *Theoretical Computer Science*, 191(1):37–59, 1998.

[122] A. N. Siegel. *Combinatorial Game Theory*. American Mathematical Society, 2013.

[123] M. Sipser. *Introduction to the theory of computation*. PWS Publishing Company, 1997.

[124] R. P. Sprague. Über mathematische Kampfspiele. *The Tôhoku Mathematical Journal*, 41:438–444, 1936.

[125] P. H. Sterbenz. *Floating-Point Computation*. Prentice-Hall, 1974.

[126] J. Stolfi and L. de Figueiredo. *Self-Validated Numerical Methods and Applications*. Monograph for 21st Brazilian Mathematics Colloquium, Rio de Janeiro, Brazil, 1997.

[127] The Coq Development Team. The Coq proof assistant: Reference manual – version 8.5. Technical report, INRIA, 2016.

[128] P. Théveny. *Numerical Quality and High Performance in Interval Linear Algebra on Multi-Core Processors*. Thèse de doctorat, Université de Lyon, École Normale Supérieure de Lyon, 2014.

[129] L. N. Trefethen. Computing numerically with functions instead of numbers. *Math. Comput. Sci.*, 1(1):9–19, 2007.

[130] W. Tucker. The Lorenz attractor exists. *Comptes-Rendus de l'Académie des Sciences-Series I-Mathematics*, 328(12):1197–1202, 1999.

[131] W. Tucker. *Validated Numerics – A Short Introduction to Rigorous Computations*. Princeton University Press, 2011.

[132] A. Turing. On computable numbers, with an application to the Entscheidungsproblem. *Proceedings of the London Mathematical Society*, 42(2):230—265, 1936.

[133] K. W. Wagner. On omega-regular sets. *Information and Control*, 43(2):123–177, 1979.

[134] H. Wang. Proving theorems by pattern recognition II. *Bell System Technical Journal*, 40(1-3):1–41, 1961.

[135] B. Weiss. Subshifts of finite type and sofic systems. *Monatshefte für Mathematik*, 77:462–474, 1973.

[136] J. H. Wilkinson. Error analysis of floating-point computation. *Numer. Math.*, 2:319–340, 1960.

[137] D. Wolfe. Go endgames are PSPACE-hard. In R. J. Nowakowski, editor, *More Games of No Chance*, volume 42 of *MSRI Publications*, pages 125–136. Cambridge University Press, 2002.

Table des figures

Table des matières