Complexité d'ordre supérieur et analyse récursive

Hugo Férée

UNIVERSITÉ DE LORRAINE

École doctorale IAEM Lorraine

Complexité d'ordre supérieur et analyse récursive

THÈSE

présentée et soutenue publiquement le 10 décembre 2014

pour l'obtention du

Doctorat de l'Université de Lorraine

(mention informatique)

par

Hugo Férée

Composition du jury

Rapporteurs :	Patrick Baillot	Directeur de recherche, CNRS
	Olivier Bournez	Professeur, LIX
	Jim Royer	Professeur, Université de Syracuse
Examinateurs :	Philippe de Groote	Directeur de recherche, Inria
	Paul-André Melliès	Chargé de recherche, CNRS
Directeur de thèse :	Jean-Yves Marion	Professeur, Université de Lorraine
Codirecteur de thèse :	Mathieu Hoyrup	Chargé de recherche, Inria

Laboratoire Lorrain de Recherche en Informatique et ses Applications — UMR 7503

Loria
Laboratoire lorrain de recherche en informatique et ses applications

Remerciements

Je tiens tout d'abord à remercier mes encadrants, à savoir Jean-Yves Marion, toujours de bon conseil, qui m'a permis de faire cette thèse dans d'excellentes conditions, ainsi que Mathieu Hoyrup qui a toujours été disponible et a su m'orienter dans mes recherches et m'encourager tout en me laissant libre de mes choix.

Je suis tout particulièrement reconnaissant aux membres de mon jury qui m'ont fait l'honneur d'accepter de juger mon travail, et tout particulièrement à mes rapporteurs Patrick Baillot, Olivier Bournez et Jim Royer qui ont pris le temps de relire ce manuscrit en détail malgré des contraintes familiale, temporelle ou linguistique.

Merci à tous ceux avec qui j'ai eu le plaisir de collaborer ou simplement d'échanger autour de sujets de recherche, ainsi qu'à ceux qui m'ont encadré et permis de progresser dans l'enseignement avec sympathie et enthousiasme, à savoir entre autres, Emmanuel, Sylvain, Marion, Vincent, Alexis et Bernard.

J'ai eu la chance d'être accueilli dans l'équipe CARTE, aussi éclectique que stimulante, notamment par Emmanuel et Romain avec qui j'ai eu le plaisir de travailler et m'ont permis de faire la connaissance inattendue d'Alain et de Frank. Je remercie aussi le reste de l'équipe pour ses conseils pertinents, ses discussions intéressantes et ses débats parfois délirants mais toujours passionnants, et tout particulièrement les membres du bureau B235 Aurélien, Hubert et Thanh Dinh qui m'ont permis d'apprendre quelque chose chaque jour au détriment d'une perte mutuelle de productivité.

Sans oublier les doctorants du Loria et des laboratoires voisins pour l'ambiance de nos réunions hebdomadaires ainsi que mes amis avec qui j'ai passé des moments inoubliables et qui me rappellent quotidiennement qu'il n'y a pas que l'informatique dans la vie.

Merci à ma famille, tout spécialement à mes parents qui m'ont toujours accompagné et conforté sans jamais désespérer d'obtenir un jour une explication sur ces notions mathématico-informatiques absconses que je manipule au quotidien.

Enfin, merci à Anne-Sophie qui a su trouver je ne sais où la patience de me supporter et les moyens de me soutenir toutes ces années.

Sommaire

Introduction **1**

Analyse récursive **5**

Chapitre 1
Fonctions d'ordre 2

1.1 Calculs sur des mots infinis 8
1.1.1 Calculabilité 8
1.1.2 Topologie 9
1.1.3 Complexité 11
1.2 Calculs sur des fonctions d'ordre 1 12
1.2.1 Machines à oracle 12
1.2.2 Complexité d'ordre 2 13
1.2.3 Temps polynomial 14
1.2.4 Caractérisation implicite 16

Chapitre 2
Analyse récursive : définitions

2.1 Représentations 20
2.2 Admissibilité 23
2.3 Calculabilité et complexité 24
2.3.1 Calculabilité 25
2.3.2 Complexité 26
2.3.3 L'exemple des fonctions réelles 27

Chapitre 3
Complexité des opérateurs réels

3.1 Complexité des normes réelles 32
3.1.1 Définitions et exemples 32
3.1.2 Dépendance d'une norme en un point et points importants . 35
3.1.3 Complexité déterministe 43
3.1.4 Complexité non-déterministe 49
3.2 Unique accès à l'oracle 54
3.3 Limites de la complexité d'ordre 2 59

Complexité d'ordre supérieur **63**

Chapitre 4
Modèles de calculs d'ordre supérieur

4.1 Schémas de Kleene 66
4.2 Fonctions Kleene-Kreisel calculables 67
4.2.1 Définition de Kleene 67
4.2.2 Définition de Kreisel 69
4.3 Basic Feasible Functionals 71
4.3.1 Définition 71
4.3.2 Limites 72
4.3.3 Caractérisations 72
4.4 Machines à oracle d'ordre supérieur 73
4.5 PCF et sémantique des jeux 74
4.5.1 PCF 75
4.5.2 Sémantique des jeux 76

Chapitre 5
Sémantique des jeux et complexité

5.1 Sémantique des jeux nominale 80
5.1.1 Arènes 81
5.1.2 Parties 84
5.1.3 Stratégies 86
5.1.4 Sous-arènes 88
5.1.5 Jeux séquentiels 88
5.2 Taille et complexité des stratégies 93
5.2.1 Taille d'une stratégie 93
5.2.2 Complexité cumulée 97
5.2.3 Machines à jeux 100
5.2.4 Temps polynomial 103
5.3 Complexité des fonctions d'ordre supérieur 103
5.3.1 Arènes pour PCF 104
5.3.2 Jeux innocents 105
5.3.3 Fonctions calculables en temps polynomial d'ordre supérieur 106
5.4 Applications et développements possibles 109

Index des notations **111**

Bibliographie

Introduction

Contexte

Le domaine de la complexité usuelle, c'est-à-dire relative aux fonctions sur les mots finis, est très largement étudié et connu. De même que pour la calculabilité, il vérifie une sorte de thèse de Church-Turing, au sens où de nombreux modèles de calcul naturels induisent les mêmes classes de complexité. En particulier, la classe FPTIME des fonctions calculables en temps polynomial bénéficie de nombreuses caractérisations par des approches très différentes (algèbres de fonctions [Cob65 ; BC92], λ-calcul [LM93], interprétation de programmes [BMM05] ou logique linéaire [Gir98 ; Laf04 ; BM10]).

Cette théorie d'ordre 1, bien qu'elle permette de définir les notions de calculabilité et de complexité sur de nombreux ensembles (notamment les nombres entiers, les graphes, les matrices, etc.), est limitée aux ensembles dénombrables. C'est pourquoi elle a été étendue aux fonctions d'ordre 2 (*i.e.* ayant des fonctions d'ordre 1 en entrée), notamment grâce au modèle des machines à oracles. Plusieurs classes de complexité analogues aux classes usuelles ont été définies avec succès, notamment une classe de fonctions calculables en temps polynomial [KC96].

Un des intérêts de cette extension est qu'elle permet de considérer ces notions sur des ensembles non dénombrables. Cette idée est notamment utilisée en analyse récursive (la *Type-2 Theory of Effectivity* de Weihrauch [Wei00]), où l'on peut étudier entre autres la calculabilité et la complexité [Ko91] des nombres réels et des fonctions réelles. Cette nouvelle théorie, bien que moins développée que la théorie d'ordre 1, est robuste et bénéficie elle aussi de nombreuses caractérisations, notamment par des algèbres de fonctions [BH05], des interprétations de programmes [Fér+14], ou à base d'équations différentielles (*i.e.* le modèle du GPAC [Bou+07 ; BGP12]). De plus, à l'ordre 2 apparaît une relation intrinsèque entre les propriétés calculatoires des objets et leurs propriétés mathématiques. En particulier, la calculabilité est très liée à la continuité, et la complexité à des caractéristiques analytiques comme le module de continuité. Ces liens sont tout particulièrement utiles pour caractériser des classes de complexité.

Toutefois, tout cela ne se généralise pas naturellement à tout ordre, et à partir de l'ordre 3 il n'y a plus de thèse de Church-Turing. Il existe plusieurs modèles de calculs

produisant des classes de fonctions calculables différentes [Lon00], et souvent incomparables. Cela est dû entre autres au fait qu'il existe plusieurs ensembles de fonctions d'ordre supérieur, selon que l'on considère seulement les fonctions totales ou héréditairement totales ou non, ou encore si ces fonctions doivent être séquentielles ou non. L'une des meilleures approches connues pour généraliser les fonctions calculables en temps polynomial à tout ordre est BFF [CU93], mais cette classe présente certains défauts, en particulier de ne pas inclure des fonctions intuitivement raisonnables en termes de complexité à partir de l'ordre 3.

Objectifs

Le premier objectif de cette thèse est de mieux comprendre ce que des contraintes sur la complexité d'un objet impliquent sur la « forme » de cet objet. En particulier, dans le cadre de l'analyse récursive, contraindre la complexité implique de contraindre le nombre d'appels effectués par une machine à oracle. Cela restreint la quantité d'information que la fonction a sur son entrée et implique des propriétés sur sa « forme ». De tels liens ont déjà été établis pour les nombres réels et les fonctions réelles, mais peu de travaux traitent de la complexité d'objets plus complexes comme les opérateurs réels.

Le second objectif est de parvenir à concevoir une définition de complexité sur des objets pour lesquels une telle notion n'existe pas encore. Ceci inclut les fonctions d'ordre supérieur à partir de l'ordre 3, mais pas seulement (par exemple certains espaces en analyse récursive). Une telle notion doit être pertinente, compatible avec les notions existantes, et suffisamment générale pour permettre de définir naturellement des classes de complexité en temps déterministe polynomial, mais aussi en temps non-déterministe ou encore en espace. Contrairement aux fonctions d'ordre 1 et 2 dont la complexité peut être définie comme un temps de calcul mesuré relativement à une borne sur la taille des entrées (d'ordre 0 ou 1), une notion pertinente de taille n'existe pas pour les entrées d'ordre 2. C'est donc l'un des principaux obstacles qu'il faut franchir si l'on souhaite définir la complexité de manière analogue aux notions usuelles.

Organisation de la thèse

Cette thèse sera structurée en deux parties, la première traitant du domaine de l'analyse récursive. Celle-ci étant basée sur des représentations par des fonctions d'ordre 1, les définitions de calculabilité et de complexité de telles fonctions seront rappelées dans le chapitre 1. Le chapitre 2 présentera les définitions et les résultats relatifs à l'analyse récursive qui seront utiles dans le chapitre suivant, notamment concernant la complexité des nombres réels et fonctions réelles. Nous mettrons tout particulièrement en évidence les résultats illustrant les liens entre la complexité et les

propriétés des objets calculés. Dans le chapitre 3 nous aborderons le domaine encore peu étudié de la complexité des opérateurs réels (*i.e.* des fonctions calculant sur des fonctions réelles). Nous examinerons tout d'abord le cas des normes réelles, pour lesquelles nous définissons des propriétés analytiques qui nous permettront de caractériser les normes calculables en temps polynomial déterministe (puis non-déterministe) relativement à un oracle (autrement dit ayant une complexité d'appel polynomiale). Puis, toujours dans l'idée de déterminer la structure induite par des contraintes de complexité, nous étudierons le cas des opérateurs ne pouvant évaluer leur entrée qu'en un point unique. Enfin, nous terminerons ce chapitre en montrant que, bien qu'elles permettent d'étendre la notion de fonction calculable à un grand nombre d'espaces et la complexité d'objets tels que les nombres, fonctions et opérateurs réels, les fonctions d'ordre 2 ne permettent pas de représenter correctement des espaces « plus grands » du point de vue de la complexité. Ce résultat sera une motivation supplémentaire pour la seconde partie.

Celle-ci traitera de la complexité des fonctions d'ordre quelconque. Le chapitre 4 décrira les idées utilisées dans les modèles de calcul d'ordre supérieur les plus classiques. Cela permettra en particulier d'illustrer les difficultés rencontrées lorsque l'on souhaite définir la complexité de telles fonctions. Il abordera en particulier la classe des *Basic Feasible Functionals*, que l'on peut considérer comme un sous-ensemble minimal d'un analogue de FPTIME à tout ordre ainsi que le langage PCF, dont la caractérisation par la sémantique des jeux semble être un bon point de départ pour une telle théorie de la complexité.

Le chapitre 5 rappellera alors les bases de la sémantique des jeux, qui permet en particulier de représenter des fonctions s'appliquant les unes aux autres comme des stratégies s'affrontant dans un jeu par l'intermédiaire d'un dialogue (*i.e.* d'un échange de questions et de réponses). Nous nous intéresserons plus particulièrement à un sous-ensemble de tels jeux, dits séquentiels, où toute interaction asynchrone est interdite. Nous définirons alors la taille de telles stratégies à partir de la taille des parties qu'elles jouent avec leurs adversaires. Intuitivement, si ces stratégies représentent des fonctions d'ordre supérieur, cela revient à mesurer la quantité d'information qu'une telle fonction a besoin de connaître sur son entrée pour fournir un résultat (la taille d'une fonction F étant une fonction de même type que F). La notion de taille étant l'un des principaux composants manquants, nous en déduisons alors une notion relativement naturelle de complexité pour de telles stratégies, ainsi qu'une classe de complexité polynomiale. Nous appliquons enfin cette théorie aux stratégies représentant des fonctions de PCF et en déduisons alors une classe de fonctions polynomiales d'ordre supérieur, qui présente certaines des caractéristiques que l'on peut attendre d'une telle classe.

Excepté le chapitre 1 qui définit des bases utiles aux autres chapitres, les deux parties sont indépendantes. Le chapitre 5 pourra cependant évoquer certains des résultats de la première partie.

Publications

— [Fér+10] : *"Interpretation of Stream Programs : Characterizing Type 2 Polynomial Time Complexity"*, travail préliminaire sur une caractérisation par interprétation de programmes de classes de complexité d'ordre 2. En collaboration avec Emmanuel Hainry, Mathieu Hoyrup et Romain Péchoux, publié dans *International Symposium on Algorithms and Computation* (ISAAC 2010).

— [FH13] : *"Higher-order complexity in analysis"*, prouve un résultat montrant les limites des interprétations par des fonctions d'ordre 1 (cf. section 3.3) et propose une solution à partir de sémantique des jeux, fournissant une base au chapitre 5. Avec Mathieu Hoyrup, publié dans *Computability and Complexity in Analysis* (CCA 2013).

— [FHG13] : *"On the Query Complexity of Real Functionals"*, étudie la complexité des opérateurs réels, en particulier des normes (cf. sections 3.1 et 3.2). En collaboration avec Walid Gomaa et Mathieu Hoyrup, publié dans *Symposium on Logic in Computer Science* (LICS 2013).

— [FGH14] : *"Analytical properties of resource-bounded real functionals "*, étend [FHG13], notamment le cas non déterministe (cf. 3.1.4). Avec Walid Gomaa et Mathieu Hoyrup, publié dans *Journal of Complexity* (JOC 2014).

— [Fér+14] : *"Characterizing polynomial time complexity of stream programs using interpretations"*, étend [Fér+10] en faisant notamment le lien avec l'analyse récursive. En collaboration avec Emmanuel Hainry, Mathieu Hoyrup et Romain Péchoux à paraître dans *Theoretical Computer Science* (TCS 2014).

Analyse récursive

Chapitre 1

Fonctions d'ordre 2

Dans ce chapitre nous décrirons les notions de calculabilité et de complexité relatives aux fonctions d'ordre 2. Elles permettront de définir les bases de l'analyse récursive dans le chapitre 2 et sont un premier pas vers une théorie de la complexité à tout ordre.

Définition 1.1. *Dans la suite nous considérerons principalement des* **types finis** *:*

$$\tau := \mathbb{N} \mid \tau \times \tau \mid \tau \to \mathbb{N}.$$

*On définit alors l'***ordre** *d'une fonction en fonction de son type τ :*
- $\mathbb{N}$ *correspond à l'ordre* 0*,*
- *si τ et τ' correspondent respectivement aux ordres n et n', alors le type produit $\tau \times \tau'$ correspond à l'ordre* $\max(n, n')$
- *enfin, si τ correspond à l'ordre n, alors $\tau \to \mathbb{N}$ correspond à l'ordre $n + 1$.*

Remarque 1.1. *Il est à noter que pour des raisons de simplicité, le type de retour des fonctions est toujours $\mathbb{N}$. Il est possible de considérer les types de la forme $\tau := \mathbb{N} \mid \tau \to \tau$, mais l'on pourra se ramener aux types précédents par décurryfication.*

Contrairement aux fonctions d'ordre 1, les domaines d'entrée des fonctions d'ordre 2 sont indénombrables. Alors qu'une entrée d'ordre 0 peut être représentée par un mot fini, le problème de la représentation d'entrées d'ordre 1 et plus n'est pas trivial.

Cela est d'autant moins évident lorsque l'on souhaite définir une notion raisonnable de complexité. En effet, l'une des définitions les plus courantes de la complexité est : le temps de calcul d'une machine en fonction de la taille de ses entrées. Si l'on souhaite faire de même pour définir la complexité à l'ordre 2, il s'agit donc aussi de choisir une représentation des entrées d'ordre 1 pour laquelle il existe une notion de taille raisonnable.

La section 1.1 décrira une première tentative pour définir la calculabilité et la complexité d'ordre 2 en représentant les entrées par des mots infinis. La section 1.2 détaillera ensuite un modèle plus adapté à la complexité dans lesquelles les entrées

sont représentées sous forme d'oracles. En particulier il permet de définir une classe robuste de fonctions polynomiales d'ordre 2.

1.1 Calculs sur des mots infinis

Étant donné un alphabet fini Σ on utilise habituellement des mots finis Σ^* pour représenter des donnée d'ordre 0 (des nombres entiers, des nombres rationnels, des graphes, des listes ou encore des matrices entre autres).

Pour représenter des fonctions de type $\mathbb{N} \to \mathbb{N}$ (mais aussi des nombres et fonctions réelles par exemple, comme on le verra dans le chapitre 2), une première approche consiste à écrire leurs valeurs successivement dans une suite infinie. Il convient ensuite d'adapter les machines de Turing usuelles à des entrées ou sorties infinies. C'est cette approche adoptée entre autres par Weihrauch [Wei00] qui sera décrite dans cette section.

1.1.1 Calculabilité

On dénotera par Σ^ω l'ensemble des suites infinies sur l'alphabet $\Sigma = \{0, 1\}$. Dans ce chapitre, s dénotera implicitement une suite et m un mot. Si n est un entier, on notera $\underline{n} \in \Sigma^*$ son écriture binaire. On écrira aussi $|m| \in \mathbb{N}$ pour la longueur d'un mot m et $s_{\restriction k}$ pour le préfixe de taille k de la suite s. Enfin, on utilisera des lettres minuscules, comme f, pour les fonctions d'ordre 1, des lettres majuscules, comme F, pour les fonctions d'ordre 2, et des lettres cursives, comme $\mathcal{F}$ pour les fonctions sur les suites infinies.

Remarque 1.2. *On commencera par remarquer qu'une fonction $f : \mathbb{N} \to \mathbb{N}$ peut effectivement être encodée dans une suite sur un alphabet de taille au moins deux : il suffit d'écrire successivement les valeurs de f en binaire en intercalant des zéros entre les chiffres, et des uns entre les valeurs :*

$$\underline{f(0)}_0 0 \underline{f(0)}_1 0 \dots \underline{f(0)}_{|\underline{f(0)}|-1} 1 \underline{f(1)}_0 0 \underline{f(1)}_1 0 \dots \underline{f(1)}_{|\underline{f(1)}|-1} 1 \dots \tag{1.1}$$

$\underline{f(i)}_j$ désignant le chiffre numéro j de $\underline{f(i)}$.

Définition 1.2. *Une machine de Turing sur les suites infinies est une machine avec des rubans d'entrée et de sortie semi-infinis, telle que sa tête d'écriture de sortie est à sens unique. Elle calcule une fonction $\mathcal{F} : \Sigma^\omega \to \Sigma^\omega$ si lorsque $s \in \Sigma^\omega$ est écrit sur le ruban d'entrée, la machine écrit successivement tous les chiffres de $\mathcal{F}(s)$ sur le ruban de sortie. Plus précisément, pour tout $i \in \mathbb{N}$, elle finira par écrire $\underline{f(s)}_i$ en position i sur le ruban de sortie.*

Par extension, on dira qu'une fonction $F : (\mathbb{N} \to \mathbb{N}) \to (\mathbb{N} \to \mathbb{N})$ est calculée par une telle machine si la fonction partielle $\mathcal{F} : \Sigma^\omega \to \Sigma^\omega$, qui associe à l'encodage d'une fonction $f : \mathbb{N} \to \mathbb{N}$ (tel que défini dans la remarque 1.2) l'encodage de son image $F(f)$, l'est.

Notons qu'une telle machine ne s'arrête pas, mais est productive dans le sens où elle finit toujours par écrire une portion supplémentaire de la sortie. De plus, le ruban de sortie est à sens unique pour qu'une valeur précédemment écrite ne puisse être effacée. Ainsi, pour obtenir le $n^{\text{ième}}$ chiffre de la sortie, il suffit d'attendre suffisamment longtemps ; il finira par être inscrit sur le ruban et ne pourra plus être modifié.

Remarque 1.3. *Tout comme les machines de Turing usuelles, c'est un modèle robuste auquel on peut par exemple fournir plusieurs entrées de type Σ^* ou Σ^ω.*

De plus, une machine avec sortie infinie peut être simulée par une machine à sortie finie. Par exemple, une machine calculant une fonction de type $\Sigma^\omega \to \Sigma^\omega$ peut être simulée par une machine calculant sa version « décurryfiée », i.e. de type $\Sigma^\omega \times \Sigma^* \to \Sigma^*$. Si la seconde entrée est un mot binaire encodant l'entier n, alors cette machine simule la première jusqu'à obtenir le $n^{ième}$ chiffre et le retourne.

Inversement, on peut reconstruire la version « curryfiée » en simulant cette machine successivement pour toutes les valeurs de n et en écrivant successivement les résultats sur le ruban de sortie.

1.1.2 Topologie

Comme nous le soulignerons dans le chapitre suivant, il existe de nombreux liens entre la calculabilité d'ordre 2 et la topologie. Commençons par définir la topologie usuelle sur les suites infinies et sur les fonctions d'ordre 1.

Définition 1.3 (Topologie de Cantor)**.** *La topologie standard sur Σ^ω est engendrée par les ensembles (appelés cylindres) de la forme $[m] = \{s \in \Sigma^\omega \mid m \sqsubseteq s\}$ où m est un mot fini et $\sqsubseteq$ indique que m est un préfixe de s.*

Définition 1.4 (Topologie sur $\mathbb{N} \to \mathbb{N}$)**.** *La topologie usuelle sur $\mathbb{N}$ est la topologie discrète (tout ensemble est ouvert), toutes les fonctions de $\mathbb{N} \to \mathbb{N}$ sont donc continues.*

La topologie usuelle sur $\mathbb{N} \to \mathbb{N}$ est engendrée par les ensembles de la forme $\{f \mid f(n) = k\}$ pour n et k entiers.

Par définition, une fonction est continue si la pré-image de tout ouvert est un ouvert. Cependant, dans ce cas particulier, la caractérisation suivante pourra s'avérer plus pratique.

Proposition 1.1. *Une fonction $\mathcal{F} : \Sigma^\omega \to \Sigma^\omega$ est continue si et seulement si pour toute suite $s \in \Sigma^\omega$ et tout entier n, il existe un préfixe $s_{\restriction k}$ de s tel que le préfixe de taille n de l'image par $\mathcal{F}$ de toute suite du cylindre $[s_{\restriction k}]$ est constant :*

$$\forall s \in \Sigma^\omega, \forall n \in \mathbb{N}, \exists k \in \mathbb{N}, \forall s' \in \Sigma^\omega, s'_{\restriction k} = s_{\restriction k} \implies \mathcal{F}(s')_{\restriction n} = \mathcal{F}(s)_{\restriction n}$$

De même, une fonction $F : (\mathbb{N} \to \mathbb{N}) \to (\mathbb{N} \to \mathbb{N})$ est continue si et seulement si :

$$\forall g, \forall n, \exists k, \forall g', g(0), \dots, g(k) = g'(0), \dots, g'(k) \implies F(g)(n) = F(g')(n).$$

Preuve

Les espace topologiques Σ^ω et $\mathbb{N} \to \mathbb{N}$ sont des espaces à base dénombrable et donc séquentiels. Autrement dit, la continuité est équivalente à la continuité séquentielle, ce qui est équivalent à ces propositions. □

Ces deux caractérisations permettent en particulier de montrer que l'encodage utilisé pour les fonctions d'ordre 1 par des suites infinies préserve la propriété de continuité.

Corollaire 1.1. *Une fonction $F : (\mathbb{N} \to \mathbb{N}) \to (\mathbb{N} \to \mathbb{N})$ est continue si et seulement si la fonction correspondante (relativement à l'encodage de la remarque 1.2) de type $\Sigma^\omega \to \Sigma^\omega$ est continue.*

Intuitivement, une fonction est continue si et seulement si toute portion finie de sa sortie ne dépend que d'une portion finie de son entrée. Or c'est aussi le cas pour les machines à entrées infinies, qui ne peuvent attendre qu'un temps fini avant d'écrire quelque chose sur le ruban de sortie. Nous pouvons donc maintenant faire le lien suivant entre calculabilité et continuité.

Proposition 1.2. *Toute fonction calculable $\mathcal{F} : \Sigma^\omega \to \Sigma^\omega$ est continue.*

Preuve

En effet, étant donné une entrée s et un entier n, après avoir écrit les n premiers chiffres de la sortie, la machine n'a pu lire qu'un préfixe fini de l'entrée. Par conséquent la machine fournira les mêmes n premiers chiffres pour toute entrée qui prolonge ce préfixe de s. Autrement dit, la fonction calculée est continue, d'après la proposition 1.1. □

Le corollaire 1.1 induit alors le même résultat sur les fonctions d'ordre 2.

Corollaire 1.2. *Si $F : (\mathbb{N} \to \mathbb{N}) \to (\mathbb{N} \to \mathbb{N})$ est calculée par une machine telle que définie dans la définition 1.2, alors elle est continue.*

Exemple 1.1. *La fonction $F_0 = \lambda f.[f \stackrel{?}{=} \lambda x.0]$ qui teste l'égalité de son entrée avec la fonction nulle n'est pas calculable car elle n'est pas continue. En effet, pour tout $n \in \mathbb{N}$, il existe une fonction non nulle f_n égale à 0 sur les entiers inférieurs à n et donc $F_0(f_n) \neq F_0(\lambda x.0)$.*

Inversement, toute fonction continue n'est pas calculable. Par exemple toute fonction non calculable d'ordre 1 est aussi une fonction non calculable d'ordre 2.

Cependant, il est possible de distinguer deux types de comportements non calculables. La fonction fournie par l'exemple 1.1 n'est pas calculable simplement parce qu'elle est discontinue et non pas parce qu'elle nécessite la puissance d'une information non calculable au sens usuel. Le théorème suivant fait cette distinction en caractérisant les fonctions calculées par des machines d'ordre 2 qui peuvent accéder à un oracle d'ordre 1 quelconque (par exemple le problème de l'arrêt).

Théorème 1.1. *Une fonction d'ordre 2 est calculable relativement à un oracle si et seulement si elle est continue.*

Preuve

Il suffit de prouver ce résultat sur les fonctions de type $\Sigma^\omega \to \Sigma^\omega$. Étant donné une fonction calculable relativement à un oracle, nous pouvons réutiliser la preuve de la proposition 1.2. En effet, même avec un oracle, une machine ne peut toujours lire qu'un préfixe fini de l'entrée avant de répondre, la fonction calculée est donc toujours continue.

Inversement, pour calculer une fonction continue $\mathcal{F} : \Sigma^\omega \to \Sigma^\omega$, on définit un oracle qui énumère tous les triplets de la forme $\langle m, n, v\rangle$ où $[m]$ est un cylindre sur lequel le préfixe de taille n de $\mathcal{F}$ est constant et où son $n^{\text{ième}}$ chiffre est v [1]. On construit ensuite une machine munie de cet oracle qui lit alternativement son entrée s et l'oracle jusqu'à trouver un triplet de la forme $\langle m, n, v\rangle$ tel que $s \in [m]$ pour chaque valeur successive de n. À chaque fois, v est le $n^{\text{ième}}$ chiffre de $\mathcal{F}(s)$ et est noté sur le ruban de sortie. Comme $\mathcal{F}$ est continue, s contient nécessairement un tel triplet pour chaque valeur de n, et cette machine calcule donc bien $\mathcal{F}$. □

1.1.3 Complexité

La définition de complexité utilisée par Weihrauch [Wei00] et Ko [Ko91] s'applique aux fonctions sur les suites infinies avant d'être appliquée entre autres aux nombres et fonctions réels.

Définition 1.5. *La complexité d'une machine calculant une fonction $\mathcal{F} : \Sigma^\omega \to \Sigma^\omega$ est bornée par une fonction $t : \mathbb{N} \to \mathbb{N}$ si pour toute entrée s et tout entier n, la machine écrit le $n^{ième}$ chiffre de la sortie en un temps inférieur à $t(n)$.*

Proposition 1.3. *La complexité de toute fonction calculable totale sur Σ^ω est bornée.*

Preuve

Étant donnée une machine, pour tout entier n, la fonction qui associe à une entrée le nombre d'étapes de calcul pour obtenir le $n^{\text{ième}}$ chiffre sur le ruban de sortie est calculable et donc continue d'après la proposition 1.2. Comme de plus l'espace de Cantor Σ^ω est compact, elle est bornée pour chaque $n \in \mathbb{N}$ et la fonction qui à n associe ce maximum borne donc la complexité de la machine. □

On peut alors tenter d'étendre la définition 1.5 aux fonctions d'ordre 2 en disant qu'une telle fonction est calculable en temps T si la fonction correspondante sur les suites infinies l'est.

Exemple 1.2. *La fonction identité $\lambda f.f : (\mathbb{N} \to \mathbb{N}) \to (\mathbb{N} \to \mathbb{N})$ est calculable en temps $\mathcal{O}(n)$. En effet, la machine qui copie son ruban d'entrée sur la sortie calcule cette fonction.*

Cependant, avec cette définition, la complexité de certaines fonction simples, comme celle de l'exemple 1.3 n'est pas bien définie.

1. Cet oracle est en fait un associé de $\mathcal{F}$, comme défini dans la section 4.2.

Exemple 1.3. *La complexité de la fonction $F = \lambda f.f(f(0))$ n'est pas bornée. En effet, il existe une suite de fonctions $(f_n)_{n\in\mathbb{N}}$ (par exemple $f_n(x) = x + n$) telle que $F(f_n)$ dépend d'une partie arbitrairement grande (en fonction de n) de son entrée. Autrement dit, une machine calculant F doit nécessairement, sur l'encodage de f_n, lire les n premières valeurs de son entrée avant de pouvoir écrire le premier chiffre de la sortie. Son temps de calcul ne peut donc pas être borné puisqu'il dépend de l'entrée d'ordre 1.*

Cela s'explique par le fait qu'une fonction d'ordre 2 est représentée par une fonction partielle sur les suites. Son domaine de définition (défini par les suites dont la forme est donnée par l'équation (1.1) et qui exclut en particulier les suites terminant par 0^ω) n'est pas compact. Cela signifie que pour une véritable notion de complexité à l'ordre 2 il est nécessaire de définir et de prendre en compte la taille des entrées d'ordre 1.

1.2 Calculs sur des fonctions d'ordre 1

Dans cette section nous présentons un autre modèle de calcul mieux adapté aux fonctions d'ordre 2 et qui, comme on le verra dans le chapitre 5, est une première étape avant un modèle adapté aux fonctions d'ordre quelconque.

1.2.1 Machines à oracle

Définition 1.6 (Machine à oracle)**.** *Une machine de Turing à oracle est une machine de Turing usuelle possédant de plus un état et un ruban spéciaux. L'oracle est une fonction $O : \Sigma^* \to \Sigma^*$ et si la machine arrive dans l'état spécial (on dit qu'elle effectue un appel à l'oracle), l'oracle lit la valeur m du ruban spécial et y écrit la valeur $O(m)$ correspondante.*

De même que précédemment, une telle machine peut être munie de plusieurs entrées d'ordre 0 ou 1 (il y a alors un état spécial par oracle). Par la suite, les oracles (de type $\Sigma^* \to \Sigma^*$) seront implicitement dénotés par $O, O', O_1, \ldots$. On dira alors que O encode une fonction $f : \mathbb{N} \to \mathbb{N}$ si pour tout entier n, $O(\underline{n}) = \underline{f(n)}$.

En termes de calculabilité, ce modèle est équivalent au précédent :

Proposition 1.4. *Une fonctions F de type $\mathbb{N}^k \times (\mathbb{N} \to \mathbb{N})^l \to \mathbb{N}$ est calculable (selon la définition 1.2) si et seulement si il existe une machine à oracle qui sur des mots $m_1, \ldots m_k$ encodant des entiers $n_1, \ldots n_k : \mathbb{N}$ et des oracles $O_1, \ldots O_l$ représentant des fonctions $f_1, \ldots f_l : \mathbb{N} \to \mathbb{N}$, s'arrête sur un mot encodant $F(n_1, \ldots n_k, f_1, \ldots f_l)$.*

Preuve

Une machine sur les suites infinies peut simuler un appel à l'oracle en allant lire sur son ruban d'entrée la valeur correspondante. Inversement, une machine à oracle peut simuler une telle machine en écrivant successivement les valeurs successives de son oracle (encodées telles que décrit dans l'équation (1.1)) sur un ruban de travail à chaque fois qu'elle en atteint l'extrémité. □

Remarque 1.4. *Le théorème 1.1 est donc toujours valable, une fonction calculable étant en particulier continue puisque sur des entrées données, une machine à oracles ne peut effectuer qu'un nombre fini d'appels à des oracles.*

1.2.2 Complexité d'ordre 2

Bien qu'équivalent en termes de calculabilité, ce modèle diffère cependant de celui de la section 1.1 par son approche de la complexité (définie principalement par Kapron et Cook [KC96]).

Définition 1.7 (Temps de calcul d'une machine à oracle)**.** *Le coût d'une étape de calcul d'une machine à oracle est 1, sauf si c'est un appel à l'oracle. Dans ce cas, le coût de l'étape est égal à la longueur de la réponse de l'oracle (i.e. $|O(m)|$ pour la requête m à l'oracle O). Le temps de calcul d'une machine sur des entrées données est alors la somme des coûts des étapes de calcul.*

Remarque 1.5. *Le coût d'un appel à l'oracle peut être vu ici comme le coût de la communication de la machine avec son oracle. Mehlhorn [Meh76] a quant à lui initialement utilisé un coût unitaire, mais Ignjatovic et Sharma [IS04] ont montré que ces deux approches étaient équivalentes, au moins dans le cas des fonctions calculables en temps polynomial. En particulier, le théorème 1.3 est aussi valide avec un coût unitaire.*

De la même façon que la complexité d'une machine de Turing usuelle est définie par son temps de calcul en fonction de la taille de ses entrées, nous devons définir la taille des fonctions d'ordre 1 pour fournir une définition de complexité similaire à l'ordre 2. Kapron et Cook [KC96] fournissent la définition suivante.

Définition 1.8 (Taille d'une fonction d'ordre 1)**.** *Si O est une fonction de type $\Sigma^* \to \Sigma^*$, sa* **taille** $|O|_1 : \mathbb{N} \to \mathbb{N}$ *est définie par :*

$$\forall n \in \mathbb{N}, |O|_1(n) = \max_{|m| \leq n} |O(m)|. \tag{1.2}$$

Par extension, si f est une fonction de type $\mathbb{N} \to \mathbb{N}$, on notera aussi $|f|_1$ la taille de son encodage de type $\Sigma^* \to \Sigma^*$.

Remarque 1.6. *La taille d'une fonction est une borne sur la taille de la sortie en fonction de la taille de l'entrée. En particulier, la taille d'une fonction calculable est bornée par la complexité d'une machine de Turing qui la calcule.*

Définition 1.9 (Complexité des machines à oracle)**.** *La complexité d'une machine à oracle $\mathcal{M}$ est bornée par $T : (\mathbb{N} \to \mathbb{N}) \to \mathbb{N} \to \mathbb{N}$ si pour toute entrée $m : \Sigma^*$ et tout oracle $O : \Sigma^* \to \Sigma^*$, le temps de calcul de $\mathcal{M}^O(m)$ est borné par $T(|O|_1, |m|)$. Par extension, une fonction d'ordre 2 est calculable en temps T si elle est calculée par une machine à oracle de complexité T.*

On peut remarquer que le temps de calcul d'une machine à oracle borne en particulier le nombre d'appels à l'oracle et donc la quantité d'information qu'elle peut obtenir de son entrée. La complexité borne donc à la fois la puissance de calcul, *i.e.* limite la possibilité de faire un calcul d'ordre 1 intrinsèquement difficile, et la dépendance de la fonction calculée sur sa fonction d'entrée (en particulier dans le cas extrême où aucun appel à l'oracle est effectué, la fonction calculée est constante en son entrée d'ordre 1).

Pour isoler ce paramètre de complexité plus fin, on peut définir la notion de complexité relative, qui sera en particulier utile dans la section 3.1.

Définition 1.10 (Complexité d'appel)**.** *Une fonction d'ordre 2 a une* **complexité d'appel** $T : (\mathbb{N} \to \mathbb{N}) \to \mathbb{N} \to \mathbb{N}$ *si elle est calculée par une machine d'ordre 2 munie d'un oracle supplémentaire de* $\mathbb{N} \to \{0,1\}$ *qui pour toute entrée* $m : \Sigma^*$ *et tout oracle* $O : \Sigma^* \to \Sigma^*$ *termine après avoir effectué au plus* $T(|O|_1, |m|)$ *appels à l'oracle* O*.*

On notera qu'il est possible de raffiner encore les mesures de complexité en regardant spécifiquement la taille des questions posées à l'oracle et la taille des réponses obtenues.

1.2.3 Temps polynomial

Maintenant que la notion de complexité est définie, il est naturel de définir un analogue de la classe FPTIME (les fonctions d'ordre 1 calculables en temps polynomial) à l'ordre 2. Cela revient à déterminer une classe de fonctions bornant la complexité des fonctions, de même que les polynômes bornent la complexité des fonctions de FPTIME. Kapron et Cook [KC96] fournissent une définition équivalente à la suivante pour définir de telles fonctions.

Définition 1.11 (Polynômes d'ordre 2)**.** *Les* **polynômes d'ordre 2** *sont des polynômes usuels contenant des variables d'ordre 1 :*

$$P ::= c \mid X \mid P + P \mid P * P \mid Y(P_1, \dots P_n) \tag{1.3}$$

où $c \in \mathbb{N}$*,* X *est une variable d'ordre 0 et* Y *une variable d'ordre 1.*

Remarque 1.7. *Les polynômes d'ordre 2 sont clos par composition, et en particulier remplacer toutes les variables d'ordre 1 d'un polynôme d'ordre 2 par des polynômes d'ordre 1 donne un polynôme d'ordre 1.*

Exemple 1.4. *La fonction définie par* $P(Y_1, Y_2, X) = Y_1(Y_2(Y_2(X^2 + 3)), X)$ *est un polynôme d'ordre 2 de type* $(\mathbb{N} \times \mathbb{N} \to \mathbb{N}) \times (\mathbb{N} \to \mathbb{N}) \times \mathbb{N} \to \mathbb{N}$*. Sa composition avec les polynômes* P_1 *et* P_2 *respectivement définis par* $P_1(X_1, X_2) = X_1 \times X_2$ *et* $P_2(X) = X^2 + 1$ *donne* $P_3(X) = (((X^2 + 3)^2 + 1)^2 + 1) \times X$ *qui est bien un polynôme d'ordre 1.*

Au contraire, la fonction $\lambda f, x.f^x(1)$*, où* f^x *désigne* x *composées successives de* f*, n'est pas un polynôme d'ordre 2.*

On peut ainsi définir naturellement la classe des fonctions calculables en temps polynomial d'ordre 2.

Définition 1.12. *Une fonction F d'ordre 2 appartient à* $\textsc{fptime}_2$ *si elle est calculée par une machine à oracle dont la complexité est bornée par un polynôme d'ordre 2.*

En particulier, cette définition est compatible avec la définition usuelle de temps polynomial d'ordre 1.

Proposition 1.5. *La restriction de* $\textsc{fptime}_2$ *aux fonctions d'ordre 1 est* fptime.

Preuve

Les machines à oracle sans entrées d'ordre 1 étant simplement des machines de Turing usuelles, et les polynômes d'ordre 2 sans variables d'ordre 1 étant des polynômes usuels, ce résultat est immédiat. □

Exemple 1.5. *La fonction application* $\mathrm{Ap} = \lambda f x.f(x)$ *de type* $(\mathbb{N} \to \mathbb{N}) \times \mathbb{N} \to \mathbb{N}$ *est calculable en temps polynomial. En effet, elle est calculée par la machine qui sur l'entrée w et l'oracle f effectue un appel à l'oracle sur w et retourne la valeur obtenue. Cette machine a une complexité de l'ordre de $\lambda f x.x + f(x)$, qui est bien un polynôme d'ordre 2.*

De même tout polynôme d'ordre 2 est calculable en temps polynomial.

Cependant, la fonction taille $|\cdot|_1$ sur les fonctions n'est pas calculable en temps polynomial, contrairement à la fonction taille sur les entiers (qui donne la taille de l'encodage binaire de l'entier).

Proposition 1.6. *La fonction $\lambda f x.|f|_1(|x|)$ n'est pas calculable en temps polynomial.*

Preuve

Intuitivement, $|f|_1(n)$ dépend de la valeur de la fonction f en environ 2^n points (plus précisément $\#\{w \in \Sigma^* \mid |w| \leq n\}$) et ne peut donc être calculé en temps polynomial. Plus formellement, supposons qu'une machine calcule cette fonction en temps P, où P est un polynôme d'ordre 2. Sur la fonction constante $\lambda x.0$ et tout entier n, elle s'arrêterait en un temps inférieur à $P(\lambda x.1, |n|)$ car $\lambda x.1$ borne la taille de $\lambda x.0$. Or d'après la remarque 1.7, ce terme est un polynôme en $|n|$. Donc pour n suffisamment grand, $P(\lambda x.1, |n|) < 2^{|n|}$. Sur ces entrées, la machine ne peut alors effectuer qu'au plus $2^{|n|} - 1$ appels à l'oracle et il existe un entier $k < 2^{|n|}$ sur lequel elle n'a pas effectué d'appel à l'oracle. Soit f_k la fonction qui vaut 4 en k et 0 sinon. Le résultat de la machine est donc identique sur f_k et $\lambda x.0$. Or $|f_k|_1(|n|) = 2 \neq |\lambda x.0|_1(|n|)$ (car $|4| = 2$ et $|k| \leq |n|$), ce qui fournit une contradiction. □

Remarque 1.8. *La conséquence directe de cette proposition est que la composition d'un polynôme d'ordre 2 avec les fonctions de taille sur les entrées (d'ordre 1 et 2) n'est en général pas calculable en temps polynomial. Ce qui signifie que, contrairement aux fonctions d'ordre 1, même si une machine à oracle $\mathcal{M}$ fonctionne en temps polynomial P, il n'est pas possible de calculer en temps polynomial à partir de ses entrées f, n la borne $P(|f|_1, |n|)$ sur son temps de calcul. Cela est cependant possible à partir de la* taille *de ses entrées d'après l'exemple 1.5.*

1.2.4 Caractérisation implicite

Cobham [Cob65] caractérise la classe de complexité FPTIME comme une algèbre de fonctions, donc indépendamment d'un modèle de machines ou d'une notion de temps de calcul. Cette sorte de thèse de Church-Turing pour la complexité permet entre autres de justifier de la pertinence de cette classe et d'appuyer l'intuition que FPTIME décrit précisément l'ensemble des fonctions calculables en temps raisonnable. Pour cela, Cobham s'inspire des fonctions récursives primitives en contrôlant le schéma de récursion.

Dans la suite, $\vec{f}$ désignera une liste finie de la forme $f_1, \dots f_n$.

Définition 1.13. *On dit que f est définie par* **composition** *(notée $\circ$) à partir de $\vec{g}$ et h si :*

$$\forall \vec{x}, f(\vec{x}) = h(\vec{g}(\vec{x}))$$

On dit que f est définie par **récursion bornée** *(Bounded Recurison on Notation, notée $\mathcal{R}_B$) à partir de g, h_0, h_1 et b si pour tout x et $\vec{y}$:*

$$\begin{cases} f(0, \vec{y}) = g(\vec{y}), \\ f(2x, \vec{y}) = h_0(x, \vec{y}, f(x, \vec{y})), \\ f(2x+1, \vec{y}) = h_1(x, \vec{y}, f(x, \vec{y})), \\ |f(x, \vec{y})| \leq |b(x, \vec{y})|. \end{cases} \tag{1.4}$$

La dernière condition de l'équation (1.4) permet de contrôler la taille de la fonction et empêcher une croissance exponentielle (si b est une fonction de taille polynomiale).

Théorème 1.2 ([Cob65])**.** *La classe* FPTIME *est engendrée par l'algèbre $[0, s_0, s_1, I, \#, \circ, \mathcal{R}_B]$, où $s_0 = \lambda x.2x$, $s_1 = \lambda x.2x+1$, $I = \lambda x.x$ et $\# = \lambda xy.2^{|x \times y|}$.*

Mehlhorn [Meh76] a plus tard défini la classe des opérateurs calculables en temps polynomial en adaptant cette algèbre aux fonctions d'ordre 2. Kapron et Cook [KC96] en fournissent une présentation semblable à la précédente :

Définition 1.14. *Une fonction F d'ordre 2 est définie par* **composition**, *notée $\circ$, à partir de $\vec{G}$ et H si :*

$$\forall \vec{f}, \vec{x}, F(\vec{f}, \vec{x}) = H(\vec{f}, \vec{G}(\vec{f}, \vec{x}))$$

Une fonction F d'ordre 2 est définie par **extension**, *notée η à partir de G si :*

$$\forall \vec{f}, \vec{g}, \vec{x}, \vec{y}, F(\vec{f}, \vec{g}, \vec{x}, \vec{y}) = G(\vec{f}, \vec{x})$$

Une fonction F d'ordre 2 est définie par **récursion bornée d'ordre 2** *(notée $\mathcal{R}_B^2$) à partir de G, H et B si pour tout $\vec{f}$, x et $\vec{y}$:*

$$\begin{cases} F(\vec{f}, 0, \vec{y}) = G(\vec{f}, \vec{y}), \\ F(\vec{f}, x, \vec{y}) = H(\vec{f}, x, \vec{y}, F(\vec{f}, \lfloor \frac{x}{2} \rfloor, \vec{y})), \\ |F(\vec{f}, x, \vec{y})| \leq |B(x, \vec{y})|. \end{cases} \tag{1.5}$$

Ces opérateurs sont principalement l'adaptation aux fonctions d'ordre 2 des opérateurs définis par Cobham.

Définition 1.15. *La classe de fonctions d'ordre 2 générée par l'algèbre* $[\textsc{fptime}, \mathrm{Ap}, \circ, \eta, \mathcal{R}_B^2]$ *est appelée* $\textsc{bff}_2$ *(pour Basic Feasible Functionals).*

Remarque 1.9. *Contrairement à celle du théorème 1.2, cette algèbre ne nécessite pas de constantes telles que* s_0, s_1 *ou* # *puisqu'elles se trouvent déjà dans* FPTIME*. En outre, l'opérateur* $\mathcal{R}_B$ *peut être construit à partir de* $\mathcal{R}_B^2$ *en le composant avec une fonction de parité.*

Tout comme pour l'algèbre de Cobham, ces opérateurs sont intuitivement raisonnables en termes de complexité. Ils permettent à leur tour de caractériser les fonctions d'ordre 2 calculables en temps polynomial par machines à oracle :

Théorème 1.3 ([KC96])**.** *La classe* $\textsc{fptime}_2$ *coïncide avec* $\textsc{bff}_2$*.*

Cette caractérisation est un argument supplémentaire pour faire de $\textsc{fptime}_2$ l'analogue à l'ordre 2 de FPTIME. D'autres arguments on été fournis dans ce but, notamment par Seth [Set93], qui montre que cette classe est la plus large qui soit stable par application et λ-abstraction et qui préserve certaines classes de complexité « naturelles » d'ordre 1.

Cette algèbre de fonctions a en fait été généralisée à tout ordre par Cook et Urquhart [CU93]. Ce système (PV^ω), étudié dans la section 4.3, chapitre 4 est à la base de la classe de complexité BFF qui généralise $\textsc{bff}_2$ à tout ordre.

Chapitre 2

Analyse récursive : définitions

L'approche usuelle du calcul, c'est-à-dire sur les mots finis, permet de manipuler des objets finis comme les entiers et plus généralement toutes les structures dénombrables. Ce n'est cependant pas le cas des ensembles indénombrables, dont l'exemple le plus commun est celui des nombres réels, déjà abordé par Turing dans son article définissant les machines éponymes [Tur36].

Les modèles usuels ne suffisent pas à définir formellement les fonctions réelles calculables mais il existe plusieurs réponses très différentes à ce problème. Par exemple le modèle BSS défini par Blum, Shub et Smale [BSS89 ; BSS89] est une machine à registres qui autorise un certain nombre d'opérations primitives comme des opérations arithmétiques (addition, multiplication, division) ou des relations binaires (égalité, relation d'ordre usuelle). Contrairement aux autres, ce modèle implique entre autres la décidabilité de l'égalité réelle, ce qui n'est pas intuitivement ni physiquement réalisable.

Le GPAC (*General Purpose Analog Computer*), introduit par Shanon [Sha41] et développé notamment par Bournez et al. [Bou+07] est un autre modèle de calcul sur les nombres réels. Il peut être décrit comme un système de circuits dont les portes sont les fonctions d'addition, de multiplication, d'intégration ou des constantes, ou de façon équivalente comme l'ensemble des systèmes d'équations différentielles du premier ordre à second membre polynomial. C'est un modèle raisonnable puisqu'il modélise des calculateurs (physiques ou électriques) existant et il permet de plus de définir une théorie de la complexité pour les fonctions réelles [BGP11 ; BGP12].

Le modèle auquel nous allons nous intéresser dans la suite est celui développé principalement par Weihrauch [Wei00] appelé TTE (*Type-2 Theory of Effectivity*). Concernant les fonctions réelles, il peut être vu comme équivalent au GPAC en termes de calculabilité et de complexité [BGP13] bien que ces deux modèles soient très différents. TTE permet cependant de définir de façon uniforme ces notions sur de nombreux ensembles, par exemple les fonctionnelles réelles, les ensembles compacts de $\mathbb{R}^n$ et plus généralement tout espace topologique à base dénombrable. En effet, cette approche

consiste à s'appuyer sur les notions de calculabilité et de complexité des fonctions d'ordre 2 (définies dans le chapitre 1) en représentant les éléments des ensembles d'entrée et de sortie par des suites infinies dans l'approche de Weihrauch, ou des fonctions d'ordre 1 dans les formalismes de Ko [Ko91] et de Kawamura et al. [KC10 ; Kaw+12b].

Ce chapitre fournira les définitions de calculabilité et de complexité dans le cadre de l'analyse récursive, basées sur celles définies dans le chapitre 1.

2.1 Représentations

Dans cette section nous détaillerons les définitions et résultats liés aux représentations, c'est-à-dire aux fonctions premettant de faire le lien entre un ensemble mathématique quelconque et l'ensemble des fonctions d'ordre 1.

Dans la suite, $f : A \hookrightarrow B$ indiquera que f est une fonction partielle de A dans B. On notera $\mathrm{dom}(f) \subseteq A$ son domaine de définition.

Définition 2.1 (Représentation)**.** *Une* **notation** *de M est une fonction surjective $\delta : \Sigma^* \hookrightarrow M$. Une* **représentation** *de M est une fonction surjective $\delta : (\Sigma^* \to \Sigma^*) \hookrightarrow M$. Dans ces cas, on appelera Σ^* et $(\Sigma^* \to \Sigma^*)$ des* **espaces de représentations** *que l'on notera indistinctement Y par la suite. De plus, si $\delta(u) = x$, on dira que u est un* **nom** *pour x.*

Initialement, c'est Σ^ω qui était utilisé comme espace de représentation par Weihrauch [Wei00], contrairement à d'autres comme Ko [Ko91] ou Kawamura et Cook [KC10] qui ont préféré utiliser $\Sigma^* \to \Sigma^*$, qui comme on l'a vu dans le précédent chapitre, est plus adapté au cadre de la complexité.

Le concept de notation n'est pas nouveau puisqu'on l'utilise implicitement à chaque fois que l'on calcule sur un ensemble dénombrable quelconque. Par exemple lorsque l'on dit qu'un ordinateur effectue des calculs sur des graphes, des matrices ou simplement des entiers, on sous-entend qu'il manipule en réalité un encodage via une notation sur les mots finis.

Par définition, des espaces indénombrables ne peuvent avoir de notation, c'est pourquoi $\Sigma^* \to \Sigma^*$ (ou Σ^ω) est nécessaire comme espace de représentation et justifie les notions de calculabilité et de complexité d'ordre 2 développées dans le chapitre précédent.

Exemple 2.1. *On notera $\nu_\mathbb{Q}$ la notation suivante des nombres rationnels : $\nu_\mathbb{Q}(m) = q \in \mathbb{Q}$ si $m = \langle a, b\rangle$ encode une paire d'entiers (a, b) telle que $q = \frac{a}{b}$.*

On pourra aussi utiliser les nombres rationnels dyadiques $\mathbb{D} = \bigcup_n \mathbb{D}_n$ avec $\mathbb{D}_n = \{\frac{p}{2^n} : p \in \mathbb{N}, 0 \leq p \leq 2^n\}$ qui ont l'avantage de s'encoder simplement : $\nu_\mathbb{D}(m) = \frac{p}{2^n}$ si m encode en binaire l'entier p et $|m| = n$.

Le cas de l'intervalle réel $[0, 1]$ est le plus étudié en analyse récursive et on le prendra donc comme principal exemple dans ce chapitre.

Exemple 2.2. *L'ensemble des nombres réels de l'intervalle* $[0,1]$ *possède les représentations suivantes :*

- *La représentation binaire, l'une des plus naturelles consiste à représenter un réel par une expansion binaire :* $\delta_b(f) = x$ *si* $x = \sum_{i\in\mathbb{N}} f(0^i)2^i$ *avec pour tout* $i \in \mathbb{N}, f(0^i) \in \{0,1\}$. *On pourra remarquer que le* $i^{ème}$ *chiffre de l'expansion binaire de* x *est encodé dans* $f(0^i)$ *et non* $f(\underline{i})$. *Ainsi, il est nécessaire de connaître* f *en une entrée de taille* i *et non* $\lfloor \log_2(i) \rfloor$ *pour obtenir ce chiffre en position* i. *On peut aussi considérer d'autres variantes de cette représentation où l'information sera plus ou moins compressée dans* f.
- *La représentation binaire signée est semblable à la précédente :* $\delta_\pm(f) = x$ *si* $x = \sum_{i\in\mathbb{N}} f(0^i)2^i$ *où pour tout* $i \in \mathbb{N}, f(0^i)$ *encode* $-1, 0$ *ou* 1.
- *La représentation de Cauchy naïve* δ_{C_n} *représente un réel par une suite convergeant vers* x, *autrement dit :* $\delta_C(f) = x$ *si la suite* $(\nu_\mathbb{Q}(f(0^i)))_{i\in\mathbb{N}}$ *converge vers* x *(on pourra dans la suite utiliser au choix une suite de rationnels ou de dyadiques de façon équivalente).*
- *De même, les représentations* $\delta_<$ *et* $\delta_>$ *représentent respectivement un réel par une suite convergente croissante ou décroissante.*
- *La représentation de Cauchy est munie d'une vitesse de convergence :* $\delta_C(f) = x$ *si* $\forall n \in \mathbb{N}, |\nu_\mathbb{Q}(f(0^n)) - x| \leq 2^{-n}$.
- *La représentation* $\delta_{[0,1]}$, *plus tard définie comme la représentation standard des réels représente un nombre réel par l'ensemble des intervalles rationnels qui le contiennent :* $\delta_{[0,1]}(f) = x$ *si* $\{f(m) \mid m \in \Sigma^*\} = \{\langle a, b\rangle \mid x \in]\nu_\mathbb{Q}(a), \nu_\mathbb{Q}(b)[\}$.

Remarque 2.1. *Il est facile de voir que la plupart de ces représentations peuvent s'adapter à la représentations de* $\mathbb{R}$ *tout entier. En particulier, la représentation de Cauchy* δ_C *s'étend naturellement sur* $\mathbb{R}$ *et la taille de la représentation d'un réel est liée à la valeur absolue du réel lui-même :*

$$\forall f, |\delta_C(f)| \leq |\nu_\mathbb{Q}(f(0))| + 1 \leq 2^{|f(0)|} + 1. \tag{2.1}$$

Étant donné une représentation pour ses ensembles de départ et d'arrivée, on peut maintenant assimiler une fonction entre des espaces quelconques à une fonction sur les espaces de représentation.

Définition 2.2 (Réalisation)**.** *Une fonction* $f : M \to M'$ *est réalisée (relativement à deux représentations* δ *et* δ' *de* M *et* M' *respectivement) par une fonction* $g : Y \to Y'$ *(où* Y, Y' *sont les ensembles représentant* M *et* M'*) si* $f \circ \delta = \delta' \circ g$ *sur* $\mathrm{dom}(\delta)$.

Remarque 2.2. *Il est possible d'adapter cette définition à des fonctions d'arité quelconque, étant donné des représentations pour chaque espace d'entrée, mais nous nous restreindront au cas d'arité 1 pour une question de simplicité.*

Comme illustré dans l'exemple 2.2, il existe de nombreuses représentations pour un même ensemble. Il s'agit donc de les comparer pour déterminer s'il en existe une plus adaptée que les autres d'un point de vue topologique.

$$\begin{array}{ccc} M & \xrightarrow{f} & M' \\ \uparrow_\delta & & \uparrow_{\delta'} \\ Y & \xrightarrow{g} & Y' \end{array}$$

FIGURE 2.1 – g réalise f.

Définition 2.3 (Réductibilité et équivalence topologique)**.** *Soient $\delta, \delta' : Y \hookrightarrow M$ des représentations.*

- **δ est continûment réductible à δ' ($\delta \leq_t \delta'$)** *s'il existe une fonction continue $F : Y \to Y$ telle que $\delta = \delta' \circ F$ sur $\mathrm{dom}(\delta)$.*
- **δ est continûment équivalente à δ' ($\delta \equiv_t \delta'$)** *si $\delta \leq_t \delta'$ et $\delta' \leq_t \delta$.*

Informellement, $\delta \leq_t \delta'$ signifie qu'un nom pour δ contient au moins autant d'information qu'un nom pour δ'.

Exemple 2.3. *Les représentations de l'intervalle $[0,1]$ définies dans l'exemple 2.2 peuvent se comparer comme illustré dans la figure 2.2*

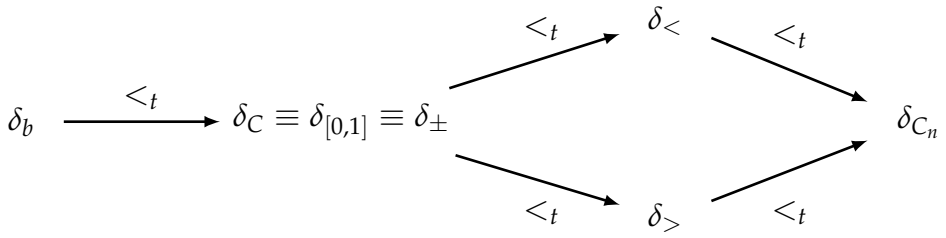

FIGURE 2.2 – Illustration des réductions entre certaines représentations de $[0,1]$.

Entre autres, $\delta_b \leq \delta_\pm$ car une représentation binaire est aussi une représentation binaire signée, donc $\delta_b = \delta_\pm \circ id$ sur $\mathrm{dom}(\delta_b)$. En revanche, il n'y a pas de réduction continue remplaçant une expansion binaire signée par une expansion binaire. En effet, une telle fonction devrait potentiellement connaître entièrement l'expansion $1, -1, -1, -1, \ldots$ pour savoir si le premier chiffre de son expansion binaire est 0 ou 1 (si un 0 ou un 1 apparaît finalement) et donc ne pourrait être continue.

La topologie finale d'une représentation est la topologie qu'elle induit sur l'espace représenté.

Définition 2.4 (Topologie finale)**.** *La topologie finale τ_δ d'une représentation δ est l'ensemble des parties X de M telles que $\delta^{-1}(X)$ est un ouvert de l'espace des représentations.*

Une bonne représentation doit préserver la topologie de l'espace qu'elle représente. Ainsi, les fonctions continues seront réalisées par des fonctions continues de l'espace de représentation.

2.2 Admissibilité

Il s'agit maintenant de déterminer les bonnes représentations, *i.e.* celles qui préserveront certaines propriétés de l'espace représenté, en particulier dont la topologie finale est celle de l'espace représenté. Pour certains espaces topologiques il est possible de définir une représentation canonique qui vérifie cette condition.

Définition 2.5 (Espace T_0)**.** *Un espace topologique (M, τ) est T_0 si tout élément de M est caractérisé par l'ensemble de ses voisinages (pour la topologie τ). Autrement dit, si deux éléments sont distincts, alors il existe un ouvert qui contient exactement l'un d'entre eux. (M, τ) est de plus à base dénombrable si la topologie τ est engendrée par une famille dénombrable d'ouverts.*

Pour définir une représentation standard, il est nécessaire d'avoir plus de structure sur de tels espaces.

Définition 2.6 (Espace topologique effectif)**.** *$S = (M, \sigma, \nu)$ est un espace topologique effectif si σ est un ensemble dénombrable de sous-ensembles de M et ν est une notation pour σ. De plus, tout élément de M est caractérisé par l'ensemble des éléments de σ auxquels il appartient. La topologie associée à S (notée τ_S) est la topologie induite par σ (autrement dit, σ est une pré-base de τ_S). En particulier, (M, τ_S) est un espace T_0 à base dénombrable. Inversement, tout espace T_0 à base dénombrable est un espace topologique effectif (une fois une base dénombrable et une notation choisie).*

Exemple 2.4. *L'intervalle réel $[0, 1]$ muni de l'ensemble des intervalles rationnels et d'une notation pour les paires de rationnels (comme définie dans l'exemple 2.1) est un espace topologique effectif.*

Étant donné un espace topologique effectif, on peut en définir une représentation standard.

Définition 2.7 (Représentation standard)**.** *Si $S = (M, \sigma, \nu)$ est un espace topologique effectif, alors sa représentation standard $\delta_S : Y \hookrightarrow M$ est définie ainsi : $\delta_S(u) = x$ si u encode la liste dans le désordre des éléments de σ contenant x. Plus précisément, u encode une suite de mots, chacun représentant (via la notation ν) un élément de σ qui contient x et ils sont tous énumérés.*

Les propriétés suivantes sont prouvées dans [Wei00].

Propriété 2.1 (Propriétés de la représentation standard)**.** *Soit $S = (M, \sigma, \nu)$ un espace topologie effectif.*

1. *δ_S est continue pour la topologie τ_S.*
2. *δ_S est ouverte (l'image d'un ouvert est un ouvert).*
3. *τ_S est la topologie finale de δ_S.*
4. *Si $\delta : Y \hookrightarrow M$ est continue (pour la topologie τ_S), alors $\delta \leq_t \delta_S$.*

5. *Si $f : M \hookrightarrow M'$ (pour un espace topologique (M', τ')), alors $f \circ \delta_S$ est continue $\iff$ f est continue.*
6. *Si deux espaces topologiques effectifs engendrent la même topologie, alors leurs représentations standard sont continûment équivalentes.*

Ces propriétés sont aussi partagées par toutes les représentations continûment équivalentes à la représentation standard. On définit donc ainsi les représentations admissibles, *i.e.* celles qui sont pertinentes d'un point de vue topologique.

Définition 2.8 (Représentation admissible)**.** *Une* **représentation admissible** *d'un espace T_0 à base dénombrable est une représentation continûment équivalente aux représentations standard de cet espace.*

Exemple 2.5. *Parmi les représentations de $[0,1]$ définies dans l'exemple 2.2, les représentations admissibles sont $\delta_\pm, \delta_C$ et $\delta_{[0,1]}$.*

Les représentations admissibles réalisent des fonctions continues par des fonctions continues.

Proposition 2.1. *$f : M \to M'$ est continue (relativement aux topologies sur M et M') si et seulement si elle est réalisée par une fonction continue pour des représentations admissibles de M et M'.*

Exemple 2.6. *Par exemple, la fonction de $[0,1] \to [0,1]$ définie par $\lambda x.\frac{3}{4}x$ est continue pour la topologie usuelle des réels, et donc continûment réalisable pour toute représentation admissible de $[0,1]$. Au contraire, elle n'est pas continûment réalisable pour la représentation δ_b par un argument similaire à celui de l'exemple 2.3.*

La représentation standard est donc la représentation la plus générale (pour $\leq_t$) qui préserve la topologie et la notion de continuité.

Théorème 2.1 (Caractérisation des représentations admissibles [Wei00])**.** *Une représentation δ est admissible pour (M, τ) si et seulement si elle est continue (pour la topologie τ) et $\delta' \leq_t \delta$ pour toute représentation continue δ' de M.*

2.3 Calculabilité et complexité

Nous avons distingué les représentations d'un même espace en fonction de leurs propriétés topologiques, nous allons maintenant les discriminer en termes de calculabilité et de complexité et ainsi pouvoir définir ces notions sur les espaces représentés à partir de celles définies sur les espaces de représentation (notamment $\Sigma^* \to \Sigma^*$ étudié dans le chapitre 1).

2.3.1 Calculabilité

Définition 2.9 (Éléments calculables)**.** *Un élément x de X est calculable relativement à une représentation $\delta : (\Sigma^* \to \Sigma^*) \hookrightarrow X$ s'il existe $f : \Sigma^* \to \Sigma^*$ calculable tel que $\delta(f) = x$.*

Exemple 2.7. *La plupart des constantes mathématiques usuelles $(\pi, e, \dots)$ sont calculables relativement à la plupart des représentations usuelles des réels. Au contraire, si $f : \mathbb{N} \to \{0,1\}$ est un problème indécidable, alors $\sum_{n\in\mathbb{N}} f(n)2^{-n}$ n'est pas calculable pour δ_b ou $\delta_\pm$ par exemple.*

De même que l'on a comparé les représentations d'un point de vue topologique, on peut les comparer en termes de calculabilité.

Définition 2.10 (Réductibilité et équivalence calculable)**.** *Soient $\delta, \delta' : Y \hookrightarrow M$ des représentations.*

- *δ* **est récursivement réductible à** *δ'* **($\delta \leq_c \delta'$)** *s'il existe une fonction calculable $F : Y \to Y$ telle que $\delta = \delta' \circ F$ sur $\mathrm{dom}(\delta)$.*
- *δ* **est récursivement équivalente à** *δ'* **($\delta \equiv_c \delta'$)** *si $\delta \leq_c \delta'$ et $\delta' \leq_c \delta$.*

Cette relation d'ordre est plus fine que $\leq_t$ puisque toute fonction calculable est continue. Elle permet notamment de comparer les notions de calculabilité induites.

Proposition 2.2. *Si $\delta \leq_c \delta'$ est x est calculable relativement à δ, alors x est calculable relativement à δ'.*

Remarque 2.3. *Évidemment, deux représentations calculablement équivalentes définissent le même ensemble d'éléments calculables. L'implication inverse n'est pas nécessairement vraie. Par exemple $\delta_\pm \not\leq_c \delta_b$ bien que ces représentations définissent le même ensemble de réels calculables.*

Définition 2.11 (Fonction calculable)**.** *Étant données deux représentations δ et δ' de M et M', $F : M \to \mathcal{M}'$ est calculable relativement à δ et δ' si elle est réalisée par une fonction calculable.*

Proposition 2.3. *Si $\delta_1 \leq_c \delta_2$ sont des représentations de M et $\delta'_1 \leq_c \delta'_2$ des représentations de M', telles que $f : M \to M'$ est calculable relativement à δ_2 et δ'_1, alors f est aussi calculable relativement à δ_1 et δ'_2.*

Exemple 2.8. *Les représentations $\delta_\pm, \delta_C$ et $\delta_{[0,1]}$ sont récursivement équivalentes et définissent le même ensemble de fonctions calculables (en particulier la plupart des fonctions usuelles continues : les fonctions arithmétiques, trigonométriques, exponentielle ou racine).*

Remarque 2.4. *Si δ et δ' sont des représentations admissibles de M et M' respectivement, alors toute fonction calculable (relativement à ces représentations) est continue, d'après la proposition 2.1.*

2.3.2 Complexité

On peut de la même façon comparer les représentations en termes de complexité et ainsi induire la notion de complexité sur les espaces représentés à partir de celle sur les espaces de représentation.

Définition 2.12. *Un élément x de X représenté par δ est calculable en temps $t : \mathbb{N} \to \mathbb{N}$ s'il existe une fonction $f : \Sigma^* \to \Sigma^*$ calculable en temps t telle que $\delta(f) = x$.*

Exemple 2.9. *Pour la représentation δ_b, un réel est calculable en temps polynomial s'il existe un programme qui calcule pour tout $n \in \mathbb{N}$ les n premiers chiffres de son expansion binaire en temps polynomial en n. De plus, la plupart des constantes mathématiques usuelles (π, e, etc.) sont calculables en temps polynomial pour toutes les représentations usuelles de $\mathbb{R}$.*

Bien évidemment, la représentation choisie influe sur la complexité. Par exemple, si $f : \mathbb{N} \to \{0,1\}$ est calculable en temps 2^n (et non polynomial), alors le nombre réel $\sum_{i\in\mathbb{N}} f(i)2^{-i}$ n'est pas calculable en temps polynomial pour la représentation de Cauchy usuelle (avec vitesse de convergence en $n \mapsto 2^{-n}$). Cependant il est calculable en temps polynomial pour la représentation de Cauchy avec vitesse de convergence $n \mapsto \frac{1}{n}$.

Remarque 2.5. *La représentation standard, bien qu'ayant du sens d'un point de vue topologique n'en a pas nécessairement d'un point de vue complexité. Par exemple, pour la représentation standard de l'intervalle réel $[0,1]$, tout nombre calculable est calculable en temps polynomial. En effet, étant donné une machine calculant un nom pour $x \in [0,1]$, on peut construire une machine qui la simule sur ses premières entrées, et retourne une notation w pour l'intervalle rationnel $[0,1]$ si elle met trop de temps à répondre. Cette machine énumère les mêmes valeurs que la machine initiale mais retarde leur énumération en intercalant w.*

Au contraire, pour les représentations $\delta_\pm$ et δ_C, il existe une hiérarchie en temps, i.e. pour toute borne de temps $t : \mathbb{N} \to \mathbb{N}$, il existe un réel calculable au mieux en temps t.

Pour aborder le point de vue de la complexité, Kawamura et Cook [KC10] ont défini la notion de réduction suivante analogue aux deux précédentes.

Définition 2.13 (Réductibilité et équivalence polynomiale)**.** *Soient $\delta, \delta' : Y \hookrightarrow M$ des représentations.*

- *δ* **est polynomialement réductible à** *δ'* **($\delta \leq_p \delta'$)** *s'il existe une fonction calculable en temps polynomial $F : Y \to Y$ telle que $\delta = \delta' \circ F$ sur $\mathrm{dom}(\delta)$.*
- *δ* **est polynomialement équivalente à** *δ'* **($\delta \equiv_p \delta'$)** *si $\delta \leq_p \delta'$ et $\delta' \leq_p \delta$.*

Exemple 2.10. *Dans le cas de l'intervalle réel $[0,1]$, on utilise habituellement les représentations admissibles $\delta_\pm$ ou δ_C ou toute autre représentation polynomialement équivalente, δ_b n'étant pas admissible, et $\delta_{[0,1]}$ pas acceptable pour la complexité d'après la remarque 2.5. Dans la suite, on pourra parler de* représentation usuelle *de $[0,1]$ (ou de $\mathbb{R}$) pour désigner de telles représentations.*

Définition 2.14 (Complexité relative à une représentation)**.** *Étant données deux représentations δ et δ' de M et M', $F : M \to \mathcal{M}'$ est calculable en temps T relativement à δ et δ' si elle est réalisée par une fonction calculable en temps T.*

En particulier, on notera $\textsc{fptime}_{\delta,\delta'}$ *l'ensemble des fonctions calculable en temps polynomial relativement à δ et δ'. Quand il n'y aura pas d'ambiguïté sur les représentations, on notera* $\textsc{fptime}_{M\to\mathcal{M}'}$.

Propriété 2.2. *Si $\delta_1 \leq_p \delta_2$ sont des représentations de M et $\delta'_1 \leq_p \delta'_2$ des représentations de M', alors* $\textsc{fptime}_{\delta_2,\delta'_1} \subseteq \textsc{fptime}_{\delta_1,\delta'_2}$.

On notera qu'il est possible de définir d'autres notions que celle de complexité déterministe en temps. La définition de complexité en espace est aussi aisément définissable, de même que la notion de complexité en temps non-déterministe dans certains cas.

Dans le cas des nombres réels, il est possible de définir la complexité non-déterministe à partir de la représentation de Cauchy ainsi :

Définition 2.15. *Un réel $x \in [0,1]$ est calculable en temps non-déterministe t s'il existe une machine non-déterministe qui pour tout n calcule sur toutes ses branches un rationnel q tel que $q \leq x + 2^{-n}$ en temps polynomial, et il existe une branche telle que de plus $q \geq x - 2^{-n}$.*

Remarque 2.6. *Intuitivement, cette définition revient à dire dans le cadre usuel de la complexité non-déterministe, que le problème de décision « $x \geq q$ à 2^{-n} près » est dans* NP.

On peut alors définir la notion de réel calculable en temps non-déterministe polynomial $\textsc{nfptime}_{[0,1]}$.

Cette définition s'adapte de plus aux fonctions :

Définition 2.16. *Une fonction $F : M \to [0,1]$ est calculable en temps non-déterministe T relativement à une représentation δ de M s'il existe une machine non-déterministe qui pour des entrées f, n calcule sur toutes ses branches un rationnel q en temps $T(|f|_1, n)$, tel que $q \leq F(\delta(f), n) + 2^{-n}$, et il existe une branche telle que de plus $q \geq F(\delta(f), n) - 2^{-n}$.*

Remarque 2.7. *Ko [Ko82] a montré que le problème* $\textsc{fptime}_{[0,1]} \neq \textsc{nfptime}_{[0,1]}$ *se situe entre les problèmes* P $\neq$ NP *et* EXP $\neq$ NEXP*. Friedman [Fri84] a de même montré des liens similaires entre les classes de comptage et l'opérateur d'intégration.*

2.3.3 L'exemple des fonctions réelles

Après le cas bien étudié des nombres réels vient naturellement celui des fonctions continues sur l'intervalle $[0,1]$, noté $\mathcal{C}[0,1]$. De même que pour les nombres réels (et comme pour tout espace métrique effectif), $\mathcal{C}[0,1]$ possède des représentations de Cauchy. Par exemple, on peut représenter une fonction continue par une suite rapidement convergente (pour la norme infini) de fonctions linéaires par morceaux ou de polynômes à coefficients rationnels. Il est aussi possible de la définir par une

fonction qui l'approche à toute précision sur l'ensemble des nombres rationnels et d'un module de continuité qui borne sa variation (ce qui, par continuité et par densité de $\mathbb{Q}$ définit de façon unique une fonction de $\mathcal{C}[0,1]$). La représentation suivante est basée sur ce principe.

Définition 2.17 (Représentation standard des fonctions réelles)**.** *La représentation* $\delta_\square : (\Sigma^* \to \Sigma^*) \to \mathcal{C}[0,1]$ *est définie ainsi par :* $\delta(\varphi) = g$ *si il existe une fonction* $g_\mathbb{Q} : \mathbb{Q} \times \mathbb{N} \to \mathbb{Q}$ *approchant* g *sur* $\mathbb{Q}$ *et un module de continuité* $g_m : \mathbb{N} \to \mathbb{N}$ *pour* g*, autrement dit :*

$$\begin{cases} \forall q \in \mathbb{Q} \cap [0,1], \forall n \in \mathbb{N}, |g_\mathbb{Q}(q,n) - g(q)| \leq 2^{-n} \\ \forall n \in \mathbb{N}, x, y \in [0,1], |x-y| \leq 2^{-g_m(n)} \implies |g(x) - g(y)| \leq 2^{-n} \end{cases},$$

tels que φ *encode* $g_\mathbb{Q}$ *et* g_m *:*

$$\forall w, w' \in \Sigma^*, \begin{cases} \varphi(0\langle w, w'\rangle) = g_\mathbb{Q}(\nu_\mathbb{Q}(w), |w'|) \\ \varphi(1w) = g_m(|w|) \end{cases}.$$

$\mathcal{C}[0,1]$ est un espace T_0 à base dénombrable et $\delta_\square$ ainsi que les représentations de Cauchy précédemment décrites sont des représentations admissibles (pour la topologie induite par la norme infini). Ces représentations sont aussi récursivement équivalentes, mais pas polynomialement. Il s'agit donc de déterminer la meilleure représentation en termes de complexité.

Tout d'abord, ces représentations rendent la fonction application $\mathrm{Ap} : \mathcal{C}[0,1] \times [0,1] \to \mathbb{R}$ définie par $\mathrm{Ap}(f,x) = f(x)$ calculable en temps polynomial relativement aux représentations usuelles de $[0,1]$ et $\mathbb{R}$. C'est en effet un pré-requis raisonnable pour une représentation qui a du sens en termes de complexité. De plus, Kawamura et Cook on montré que $\delta_\square$ est la représentation la plus générale (pour $\leq_p$) qui rende cette fonction calculable en temps polynomial.

Théorème 2.2 ([KC10])**.** *Si* δ *est une représentation de* $\mathcal{C}[0,1]$*, alors on a* $\delta \leq_p \delta_\square$ *si et seulement si* $\mathrm{Ap} \in \textsc{fptime}_{\delta,\delta_C,\delta_\mathbb{R}}$.

Ce théorème signifie intuitivement que $\delta_\square$ est «admissible» en termes de complexité si l'on considère que Ap doit être calculable en temps polynomial.

Remarque 2.8. *On pourrait de même essayer de déterminer la représentation la plus générale qui rend tout un ensemble de fonctions calculables en temps polynomial, par exemple dans une structure algébrique où l'on souhaite que la complexité de toutes les opérations de base soit raisonnable.*

Cette représentation est d'autant plus pertinente qu'elle fait le lien avec la calculabilité et la complexité sur les nombres réels.

Proposition 2.4. *Une fonction* $f : [0,1] \to \mathbb{R}$ *est calculable (relativement aux représentations usuelles de* $[0,1]$ *et* $\mathbb{R}$*) si et seulement si c'est un élément calculable de* $\mathcal{C}[0,1]$ *pour la représentation* $\delta_\square$*. De même, une fonction* $f : [0,1] \to \mathbb{R}$ *est calculable en temps polynomial si et seulement si c'est un élément calculable en temps polynomial de* $\mathcal{C}[0,1]$ *pour la représentation* $\delta_\square$*.*

Enfin, on peut remarquer que la taille d'une représentation est liée à la «taille» de la fonction correspondante (au sens de la norme uniforme et de son module de continuité).

Proposition 2.5. *Toute fonction $f \in \mathcal{C}[0,1]$ de module de continuité m dont la norme uniforme est bornée par $M \in \mathbb{N}$ a une représentation de taille linéaire en M et m.*

Preuve

Soit f une telle fonction. Alors sur tout rationnel et pour toute précision n, elle peut être approchée par un dyadique de taille de l'ordre de $M+n$. Elle possède donc une représentation φ qui vérifie $|\varphi(0\langle w, w'\rangle)| \leq M + |w'|$ et $|\varphi(1w)| \leq m(|w|)$ pour tout $w, w' \in \Sigma^*$ et est donc bien linéaire en M et m. □

Remarque 2.9. *Il existe d'autres résultats de ce genre, faisant le lien entre complexité des réels et des fonctions réelles. En particulier, Ko [Ko82] a montré que les réels calculables en temps polynomial non-déterministe (tels que donnés par la définition 2.15) sont exactement les réels qui sont la valeur maximale d'une fonction de $\mathcal{C}[0,1]$ calculable en temps polynomial (pour $\delta_\square$), autrement dit,*

$$x \in \textsc{nfptime}_{[0,1]} \iff \exists f \in \textsc{fptime}_{\mathcal{C}[0,1]}, x = \max_{y \in [0,1]} f(y).$$

On pourra d'ailleurs faire le lien avec la caractérisation usuelle d'une fonction de NP *par une fonction polynomiale munie d'un certificat (ici c'est y qui joue le rôle du certificat).*

En particulier, calculer le max d'une fonction est difficile (cf. proposition 3.1).

La deuxième partie de la proposition 2.5 implique notamment une propriété analytique des fonctions réelles calculables en temps polynomial. En effet, si une fonction a une représentation polynomiale, c'est qu'elle a un module de continuité borné par un polynôme, d'après la définition de $\delta_\square$. Il y a donc deux raisons pour lesquelles une fonction peut ne pas être calculable en temps polynomial : soit elle encode un problème difficile (dans son approximation sur les rationnels), soit elle varie trop rapidement (et son module de continuité n'est pas polynomial). On peut donc énoncer un résultat analogue au théorème 1.1 pour décrire ce lien implicite entre complexité et module de continuité.

Théorème 2.3 ([Ko91])**.** *Une fonction de $\mathcal{C}[0,1]$ est calculable en temps polynomial relativement à un oracle si et seulement si elle a un module de continuité polynomial.*

Dans le chapitre suivant nous allons tenter de fournir un résultat similaire à un ordre de plus, c'est-à-dire à propos de la complexité des opérateurs réels (donc des fonctions définies sur $\mathcal{C}[0,1]$).

Chapitre 3

Complexité des opérateurs réels

Il existe de très nombreux résultats permettant de comprendre les calculs sur les nombres réels tels que définis dans le chapitre précédent. Nous avons vu qu'il est possible d'établir des liens entre des propriétés mathématiques propres aux objets calculés et à leurs propriétés calculatoires, notamment entre la continuité et la calculabilité, mais aussi entre des propriétés analytiques et des bornes de complexité (comme par exemple le module de continuité, dans le théorème 2.3). Cependant, peu de résultats permettent de donner une intuition des implications de la complexité des opérateurs réels, *i.e.* des fonctions de type $\mathcal{C}[0,1] \to \mathbb{R}$.

Nous allons donc maintenant fournir différents résultats permettant de mieux comprendre ces opérateurs, publiés dans [FHG13] et [FGH14]. Dans la section 3.1 nous nous intéresserons en particulier au cas des normes réelles qui possèdent de nombreuses propriétés permettant de simplifier l'étude. L'idée générale étant que le fait de borner la complexité d'une machine à oracle limite le nombre d'appels qu'elle peut effectuer (c'est la notion de complexité d'appel abordée dans la définition 1.10) limitant alors le nombre de points auxquels elle peut interroger son entrée. Nous définirons alors des outils permettant de mesurer comment cela se traduit du côté des normes, en particulier nous définirons la *dépendance* d'une norme en un point et l'ensemble des *points importants* relatifs à une norme. Nous caractériserons alors dans le théorème 3.6 les normes calculables en temps polynomial relativement à un oracle en utilisant ces outils. Nous remarquerons ensuite que ces notions peuvent être adaptées à la notion de calcul non-déterministe (abordée dans la définition 2.16), et nous obtiendrons un résultat similaire dans le théorème 3.8.

Toujours avec l'objectif de mieux comprendre les implications d'une limitation de la complexité sur la forme des objets calculés, la section 3.2 traitera du cas minimal où un opérateur réel calculé par une machine ne peut faire au plus qu'un seul appel à son oracle.

Enfin, la section 3.3 montrera que même si la théorie résumée dans le chapitre 2 permet de traiter la calculabilité et la complexité sur des ensembles tels que $\mathbb{R}$, $\mathcal{C}[0,1]$

ou $\mathcal{C}[0,1] \to \mathbb{R}$, elle atteint ses limites pour certains ensembles plus grand (par exemple $(\mathcal{C}[0,1] \to \mathbb{R}) \to \mathbb{R}$), ce qui fournira une incitation supplémentaire à définir une notion de complexité à l'ordre supérieur dans les chapitres suivants.

Dans ce chapitre, on utilisera implicitement les représentations usuelles de $[0,1]$, $\mathbb{R}$ et $\mathcal{C}[0,1]$ décrites dans le chapitre précédent (plus précisément la représentation de Cauchy et la réprésentation $\delta_\square$).

3.1 Complexité des normes réelles

3.1.1 Définitions et exemples

Si A est un sous-ensemble de $[0,1]$, alors on notera $d(\beta, A) = \inf\{|\alpha - \beta| \mid \alpha \in A\}$ la distance d'un point $\beta \in [0,1]$ à A. On appellera alors $\mathcal{B}(\alpha, r)$ le voisinage de rayon r autour de A, c'est-à-dire l'ensemble des points à distance strictement inférieure à r de A et on notera de plus $\overline{\mathcal{V}}(\alpha, r)$ le voisinage fermé correspondant.

Définition 3.1. *Si $f \in \mathcal{C}[0,1]$, on définit le **support** de f par* $\mathrm{Supp}(f) = \{x \in [0,1] \mid f(x) \neq 0\}$. *On dira que f est supporté sur $A \subseteq [0,1]$ si* $\mathrm{Supp}(f) \subseteq A$.

On désignera par la suite par $Lip_1 \subseteq \mathcal{C}[0,1]$ l'ensemble des fonctions 1-lipschitziennes de $\mathcal{C}[0,1]$, c'est-à-dire les fonctions $f \in \mathcal{C}[0,1]$ vérifiant pour tout $x, y \in [0,1]$, $|f(x) - f(y)| \leq |x - y|$, autrement dit les fonctions ayant un module de continuité borné par $n \mapsto n$.

Définition 3.2 (Norme)**.** *On rappelle que $F : \mathcal{C}[0,1] \to \mathbb{R}$ est une norme si :*
- $\forall f \in \mathcal{C}[0,1], F(f) = 0 \implies f = 0$
- $\forall f \in \mathcal{C}[0,1], \alpha \in \mathbb{R}, F(\alpha \cdot f) = |\alpha| \cdot F(f)$
- $\forall f, g \in \mathcal{C}[0,1], F(f+g) \leq F(f) + F(g)$ *(l'inégalité triangulaire).*

Exemple 3.1. *On rappelle les définitions de deux normes usuelles sur $\mathcal{C}[0,1]$ qui serviront entre autres d'exemples dans la suite.*
- *La norme uniforme (ou norme infini) :* $\|f\|_\infty = \sup\{|f(x)| \mid x \in [0,1]\}$.
- *La norme L^1 :* $\|f\|_1 = \int_0^1 |f(x)|dx$.

Cependant, ces normes, ainsi que la plupart des normes usuelles, ne sont pas calculables en temps polynomial.

Proposition 3.1 ([KF82])**.** *La norme L^1 et la norme uniforme ne sont pas calculables en temps polynomial.*

L'idée de la preuve est qu'une machine polynomiale ne peut, à précision donnée, évaluer qu'un nombre polynomial de points. On peut donc définir une fonction qui varie beaucoup entre ces points et donc dont la norme varie de même, pour laquelle la machine retournera le même résultat.

On peut cependant calculer une borne supérieure sur la norme infini d'une fonction $f \in \mathcal{C}[0,1]$ en temps polynomial (en la taille de son représentant pour $\delta_\square$).

Lemme 3.1. *Il existe une machine à oracle qui sur une représentation de $f \in \mathcal{C}[0,1]$ calcule en temps polynomial en la taille de cette représentation un entier M tel que $\|f\|_\infty \leq M$.*

PREUVE

Il suffit de demander à l'oracle d'entrée une approximation q de $f(0)$ à précision 2^0 et le module $m(0)$ de f à précision $1 = 2^{-0}$.

On a alors pour tout $x \in [0,1]$, $|f(x)| \leq |f(x) - f(0)| + |f(0)| \leq 2^{m(0)} + |q| + 1$, et donc $\|f\|_\infty \leq 2^{m(0)} + \lceil |q| \rceil q + 1$. □

Il est cependant possible de construire des normes calculables en temps polynomial.

Exemple 3.2. *Soit $(q_n)_{n\in\mathbb{N}}$ une énumération polynomiale des nombres rationnels de l'intervalle $[0,1]$ (ou de tout autre ensemble dense dans $[0,1]$), i.e. telle qu'il existe une fonction $f_q : \Sigma^* \to \Sigma^*$ calculable en temps polynomial telle que $\nu_{\mathbb{Q}}(f_q(0^n)) = q_n$ pour tout $n \in \mathbb{N}$. On définit F_q par*

$$F_q(f) = \sum_{n\in\mathbb{N}} \frac{|f(q_n)|}{2^n}.$$

Il est facile de vérifier que F_q est une norme. On peut aussi montrer que F_q est calculable en temps polynomial. En effet, pour calculer $F_q(f)$ à précision 2^{-p}, on calcule une borne 2^n sur la norme infini de f grâce au lemme 3.1 puis on calcule la somme partielle jusqu'au terme d'indice $n + p$ pour obtenir une erreur d'au plus $2^{-(p+1)}$, le reste de la somme étant inférieur à $2^{-(p+1)}$ on obtient une 2^{-p} approximation de $F_q(f)$.

Nous avons vu dans la remarque 2.7 que dans le cas des nombres réels il est aussi difficile de distinguer certaines classes de complexité que dans leurs analogues de la complexité usuelle (comme $\text{P} \stackrel{?}{=} \text{NP}$ par exemple). Au contraire, la proposition 3.1 montre que la norme uniforme n'est pas calculable en temps polynomial, alors que l'on peut dire qu'elle est dans l'analogue pour $\mathcal{C}[0,1] \to \mathbb{R}$ de la classe NP. Le fait est que les classes de complexité des fonctions ou des opérateurs réels sont plus faciles à séparer que les classes usuelles. Cela s'explique par le fait que borner le temps de calcul d'une telle fonction ne borne pas seulement sa puissance de calcul au sens usuel, mais aussi l'information qu'elle peut obtenir de ses entrées. Par exemple, une machine qui fonctionne en temps polynomial peut difficilement distinguer des fonctions pourtant très différentes si elles coïncident sur le petit ensemble d'entrées évaluées par l'oracle. Dans le cas d'une norme une contrainte sur la complexité va avoir des implications sur la topologie induite par la norme. On peut alors se demander s'il est possible de caractériser les normes polynomiales avec des propriétés purement topologiques ou analytiques.

Il existe tout d'abord une approche topologique pour comparer les normes entre elles, qui est liée à leur complexité.

Définition 3.3. *Étant donné deux normes F et G sur $\mathcal{C}[0,1]$, on dit que F est plus faible que G s'il existe une constante $c \in \mathbb{R}$ telle que pour tout $f \in \mathcal{C}[0,1], F(f) \leq c \cdot G(f)$. De façon*

équivalente, étant donné une suite $(f_n)_{n\in\mathbb{N}}$ la convergence de la suite $(G(f_n))_{n\in\mathbb{N}}$ implique la convergence de la suite $(F(F_n))_{n\in\mathbb{N}}$.

On rappelle que la convergence en probabilité est plus faible que la convergence L^1, elle-même plus faible que la convergence uniforme.

La proposition suivante, illustrée par la figure 3.1, est valable en particulier pour les normes calculables en temps polynomial.

Proposition 3.2. *Si F est une norme calculable en temps sous-exponentiel alors*

1. *la norme F est strictement plus faible que la norme uniforme ;*
2. *la norme F n'est pas comparable avec la norme L^1 ;*
3. *la convergence pour F n'est pas comparable avec la convergence en probabilité ;*
4. *la norme F n'est pas complète.*

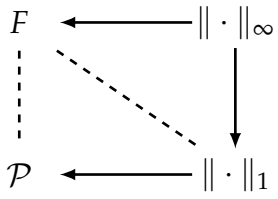

FIGURE 3.1 – Représentation graphique de la proposition 3.2. Une flèche pleine indique l'implication stricte entre deux notions de convergence ; une ligne pointillée indique leur incomparabilité.

PREUVE

Sachant que la convergence uniforme implique la convergence pour la norme L^1, qui implique la convergence en probabilité, il suffit de montrer que la convergence de F n'implique pas la convergence en probabilité et que la convergence uniforme n'implique pas la convergence pour F pour montrer les points 1, 2 et 3. On construit donc deux suites $(f_n)_{n\in\mathbb{N}}$ et $(g_n)_{n\in\mathbb{N}}$ telles que $(f_n)_{n\in\mathbb{N}}$ converge pour F mais ne converge pas en probabilité, et $(g_n)_{n\in\mathbb{N}}$ converge pour la norme L^1 mais pas pour la norme F.

Soit $\mathcal{M}$ une machine calculant F en temps sous-exponentiel et k un entier naturel. Pour une précision k et sur une entrée représentant la fonction nulle munie du module de continuité $m_k = n \mapsto n + k$ cette machine fait un ensemble A_k d'appels à l'oracle évaluant la fonction d'approximation de l'entrée. On pose $f_k(x) = \min(1, 2^k d(x, A_k))$. C'est une fonction 2^k-lipschitzienne donc de module m_k qui est nulle sur A_k. À précision k, $\mathcal{M}$ ne peut pas la distinguer de la fonction nulle (car les réponses de l'oracle sont identiques) et répond donc une notation pour un rationnel q tel que $|q| = |q - F(0)| \leq 2^{-k}$. Donc $F(f_k) \leq 2^{-(k+1)}$ et la suite $(F(f_k))_{k\in\mathbb{N}}$ converge vers 0.

On pose ensuite $g_k = 1 - f_k$. La suite $(F(g_k))_{k\in\mathbb{N}}$ ne converge donc pas pour F (car $|F(1) - F(f_k)| \leq F(g_k)$). Comme A_k est de taille sous-exponentielle $t(k)$ (car la complexité d'appel est bornée par la complexité en temps) et que sur un ensemble de mesure au plus $\#A_k 2^{-(k-1)} \leq t(k)2^{-(k-1)}$ qui converge vers 0, g_k converge vers 0 en probabilité.

Enfin, supposons que F soit complète. Alors d'après le théorème d'inversion bornée de Rudin [Rud91] la fonction identité entre les deux espaces de Banach $(\mathcal{C}[0,1], F)$ et $(\mathcal{C}[0,1], \|\cdot\|_\infty)$ est continue et la norme infinie est alors plus faible que F, ce qui contredit le point 1 et prouve le point 4. □

Ces propriétés topologiques contraignent donc l'ensemble des normes polynomiales. Nous allons maintenant restreindre d'avantage cette classe de complexité (entre autres) par des propriétés analytiques.

Comme illustré dans le preuve de la proposition 3.2, une norme polynomiale ne peut dépendre, à une précision donnée, que d'un nombre polynomial de points qui «couvrent» un ensemble de faible mesure. Ce sont ces intuitions qui vont être formalisées par les notions de *dépendance* et de *points importants*.

3.1.2 Dépendance d'une norme en un point et points importants

Nous souhaitons maintenant exprimer le fait qu'une norme sur $\mathcal{C}[0,1]$ dépend d'un point $\alpha \in [0,1]$. Pour cela nous définissons une notion quantitative qui dépend de la taille d'un voisinage de α et qui mesure si la norme change de façon non négligeable lorsque l'on modifie son entrée autour de ce point. De plus, comme nous considérons uniquement des normes, il nous est possible de nous ramener au cas où la fonction modifiée est la fonction nulle et que la modification est 1-lipschitzienne.

À partir de maintenant nous supposerons que F est une norme sur $\mathcal{C}[0,1]$ plus faible que la norme infini et on notera c_F une constante telle que pour tout $f \in \mathcal{C}[0,1]$, $F(f) \leq c_F \cdot \|f\|_\infty$.

Dépendance d'une norme en un point

Nous aurons besoin dans la suite de la fonction triangulaire $h_{\alpha,r}$ (avec $\alpha, r \in [0,1]$), qui est 1-Lipschitz et supportée sur $\mathcal{B}(\alpha, r)$ définie par $h_{\alpha,r}(x) = \max(0, r - |x - \alpha|)$.

Intuitivement, la dépendance d'une norme en un point α sera grande si pour une petite valeur de r, il existe une fonction bornée en valeur absolue par $h_{\alpha,r}$ (donc non-nulle seulement autour d'un petit voisinage de α) dont la norme est non-négligeable. Autrement dit, plus la dépendance est grande, plus la norme est sensible à une variation de son entrée autour de ce point.

Définition 3.4 (Dépendance)**.** *La* **dépendance** *d'une norme F en un point $\alpha \in [0,1]$, illustrée dans la figure 3.2, est la fonction $\mathrm{d}_{F,\alpha} : \mathbb{N} \to \mathbb{R}$ définie par :*

$$\begin{aligned}\mathrm{d}_{F,\alpha}(n) &= \sup\{l > 0 \mid \exists f \in Lip_1, |f| \leq h_{\alpha,\frac{1}{l}} \text{ et } F(f) > 2^{-n}\} \\ &= \inf\{l > 0 \mid \forall f \in Lip_1, |f| \leq h_{\alpha,\frac{1}{l}} \text{ implique } F(f) \leq 2^{-n}\}.\end{aligned}$$

Ces deux définitions sont bien équivalentes car le premier ensemble est ouvert, fermé inférieurement (donc un intervalle de la forme $]0, \mathrm{d}_{F,\alpha}(n)[$) et le second ensemble est son complémentaire dans $]0, +\infty[$). En particulier $\mathrm{d}_{F,\alpha}(n)$ est le maximum du premier ensemble.

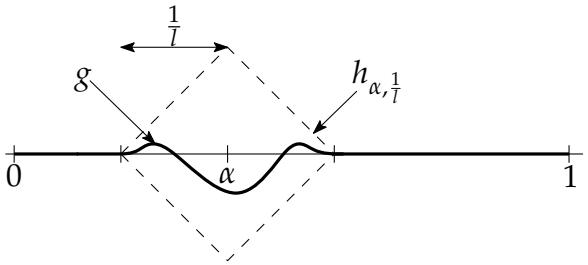

FIGURE 3.2 – Illustration de la dépendance d'une norme en un point α. La dépendance est plus grande que l en n s'il existe une telle fonction g dont la norme est plus grande que 2^{-n}.

Remarque 3.1. *Pour tout α, la suite $(\mathrm{d}_{F,\alpha}(n))_{n\in\mathbb{N}}$ est croissante, non bornée, et on a $\mathrm{d}_{F,\alpha}(n) \leq 2c_F \cdot 2^n$. De plus, l'ensemble $\{l > 0 \mid \forall f \in Lip_1, |f| \leq h_{\alpha,\frac{1}{l}}$ implique $F(f) \leq 2^{-n}\}$ est fermé, donc la borne inférieure est atteinte.*

Remarque 3.2. *Pour une norme monotone (i.e. si $|f| \geq |g|$ alors $F(f) \geq F(g)$) la dépendance en α est atteinte par $\max_{l>0} F(x \mapsto d(x, [0,1] \setminus \mathcal{B}(\alpha, \frac{1}{l})))$. En effet, comme la norme est monotone, elle est maximale en $h_{\alpha,\frac{1}{l}}$. On a donc :*

$$\mathrm{d}_{F,\alpha}(n) = \sup\left\{l \mid F(h_{\alpha,\frac{1}{l}}) > 2^{-n}\right\}.$$

Illustrons la notion de dépendance sur les normes décrites en début de chapitre.

Exemple 3.3. *La dépendance de la norme uniforme vérifie :*

$$\forall \alpha \in [0,1], n \in \mathbb{N}, 2^n \leq \mathrm{d}_{\|\cdot\|_\infty,\alpha}(n) \leq 2^{n+1}.$$

Plus précisément, en utilisant la remarque 3.2,

$$\mathrm{d}_{\|\cdot\|_\infty,\alpha}(n) = \begin{cases} 2^n & si\ 2^{-n} \leq \alpha \\ \frac{1}{\alpha} & si\ 2^{-n-1} \leq \alpha \leq 2^{-n} \\ \frac{1}{2^{-n}-\alpha} & si\ \alpha \leq 2^{-n-1}. \end{cases}$$

On a donc pour tout $\alpha, \mathrm{d}_{\|\cdot\|_\infty,\alpha}(n) = 2^n$ pour n suffisamment grand.

Donc la dépendance de la norme infini est asymptotiquement maximale, d'après la remarque 3.1.

Exemple 3.4. *La dépendance de la norme L^1, bien qu'exponentielle, est plus petite que celle de la norme uniforme.*

$$\forall \alpha \in [0,1], n \in \mathbb{N}, 2^{\frac{n-1}{2}} \leq \mathrm{d}_{\|\cdot\|_1,\alpha}(n) \leq 2^{\frac{n+1}{2}}$$

Plus précisément, $\mathrm{d}_{\|\cdot\|_1,0}(n) = \mathrm{d}_{\|\cdot\|_1,1}(n) = 2^{\frac{n-1}{2}}$, et pour tout $\alpha \in]0,1[$ et pour tout $n \in \mathbb{N}$ suffisamment grand, $\mathrm{d}_{\|\cdot\|_1,\alpha}(n) = 2^{\frac{n}{2}}$.

Exemple 3.5. *Soit F_q une norme telle que décrite dans l'exemple 3.2. Intuitivement F_q dépend beaucoup des q_i (et plus particulièrement des premiers) et éventuellement des points qui leur sont proches. En effet,pour tout $i \in \mathbb{N}$ et tout $n \geq i$, $\mathrm{d}_{F_q,q_i}(n) \geq 2^{n-i}$.*

En revanche elle dépend peu des points qui sont éloignés de l'ensemble des q_i. Plus formellement, pour presque tout point (au sens de la mesure de Lebesgue), la dépendance est bornée par un polynôme. Pour tout $\alpha \in [0,1]$ tel que pour tout $i \in \mathbb{N}$, $|\alpha - q_i| \geq \varepsilon/i^2$ on a pour tout $n \in \mathbb{N}, \mathrm{d}_{F_q,\alpha}(n) \leq n^2/\varepsilon$. En effet, si $f \in Lip_1$ est bornée en valeur absolue par $h_{\alpha,\frac{\varepsilon}{n}2}$ alors f est nulle sur $\{q_0,\ldots,g_n\}$, donc $F_q(f) = \sum_{i>n} 2^{-i}|f(q_i)| \leq 2^{-n}$ car $\|f\|_\infty \leq 1$. L'ensemble de tels α est de mesure au moins $1 - \varepsilon\pi^2/3$, ce qui tend vers 1 quand ε tend vers 0. Donc pout presque tout $\alpha \in [0,1], \mathrm{d}_{F,\alpha}$ est bornée par un polynôme (paramétré par ε).

Proposition 3.3. *Pour tout $n \in \mathbb{N}$, la fonction $\alpha \mapsto \mathrm{d}_{F,\alpha}(n)$ est continue sur $[0,1]$. De plus, pour tout $\alpha, \beta \in [0,1]$,*

$$|\mathrm{d}_{F,\alpha}(n) - \mathrm{d}_{F,\beta}(n)| \leq |\alpha - \beta|\mathrm{d}_{F,\alpha}(n)\mathrm{d}_{F,\beta}(n).$$

Preuve

Soit $l < \mathrm{d}_{F,\alpha}(n)$, $f \in Lip_1$ bornée en valeur absolue par $h_{\alpha,\frac{1}{l}}$ et vérifiant $F(f) > 2^{-n}$. On a $h_{\alpha,\frac{1}{l}} \leq h_{\beta,\frac{1}{l}+|\alpha-\beta|}$, donc $\mathrm{d}_{F,\beta}(n) \geq l/(1 + l|\alpha - \beta|)$. Comme c'est vrai pour tout $l < \mathrm{d}_{F,\alpha}(n)$, en faisant tendre l vers $\mathrm{d}_{F,\alpha}(n)$ on obtient que $\mathrm{d}_{F,\beta}(n) \geq \mathrm{d}_{F,\alpha}(n)/(1 + \mathrm{d}_{F,\alpha}(n)|\alpha - \beta|)$, autrement dit $\mathrm{d}_{F,\alpha}(n) - \mathrm{d}_{F,\beta}(n) \leq |\alpha - \beta|\mathrm{d}_{F,\alpha}(n)\mathrm{d}_{F,\beta}(n)$. Le raisonnement étant symétrique entre α et β on obtient le résultat. De plus, comme $\mathrm{d}_{F,\alpha}(n)$ et $\mathrm{d}_{F,\beta}(n)$ sont bornés (d'après la remarque 3.1) en faisant converger β vers α dans l'inégalité on obtient la continuité de $\alpha \mapsto \mathrm{d}_{F,\alpha}(n)$. □

Proposition 3.4. *Si F est une norme plus faible que G, alors il existe une constante k telle que pour tout $\alpha \in [0,1]$ et pour tout $n \in \mathbb{N}$, $\mathrm{d}_{F,\alpha}(n) \leq \mathrm{d}_{G,\alpha}(n+k)$.*

Preuve

La preuve est immédiate d'après la définition de la dépendance, avec k tel que $F \leq 2^k G$. □

En revanche, la dépendance ne permet pas nécessairement de distinguer des normes incomparables. Par exemple, si F est une norme, alors les deux normes $f \mapsto F(f) + |f(0) - f(1)|$ et $f \mapsto F(f) + |f(0)| + |f(1)|$ ont la même fonction de dépendance, mais ne sont pas a priori comparables.

Définition 3.5 (Dépendance maximale)**.** *La* **dépendance maximale** *d'une norme est la fonction $\mathrm{D}_F : \mathbb{N} \to \mathbb{R}$ définie par :*

$$\mathrm{D}_F(n) = \max_{\alpha \in [0,1]} \mathrm{d}_{F,\alpha}(n)$$

La remarque 3.1 implique que pour toute norme plus faible que la norme uniforme, il existe une constante c telle que $\mathrm{D}_F(n) \leq c \cdot 2^n$. Le résultat suivant fournit une borne inférieure, qui est atteinte par exemple par la norme L^1.

Proposition 3.5. *Pour toute norme F il existe une constante $c > 0$ telle que pour tout $n \in \mathbb{N}, \mathrm{D}_F(n) \geq c \cdot 2^{\frac{n}{2}}$.*

PREUVE

Soit $N \in \mathbb{N} \setminus \{0\}$. La somme de fonctions $\sum_{i=0}^{N} h_{\frac{i}{N},\frac{1}{N}}$ est alors égale à la fonction constante $\frac{\mathbf{1}}{N}$. D'après l'inégalité triangulaire, $\frac{F(1)}{N} \leq \sum_i F(h_{\frac{i}{N},\frac{1}{N}}) \leq (N+1)\max_i F(h_{\frac{i}{N},\frac{1}{N}})$ donc il existe $i \in \{0,\ldots,N\}$ tel que $F(h_{\frac{i}{N},\frac{1}{N}}) \geq \frac{F(1)}{N(N+1)}$. Soient $0 < c < \sqrt{F(1)}/4$ et $N = \lceil c2^{\frac{n}{2}} \rceil$. On a alors $\frac{F(1)}{N(N+1)} > \frac{F(1)}{(N+1)^2} > 2^{-n}$ pour n suffisamment grand, et donc $\mathrm{D}_F(n) \geq \mathrm{d}_{F,\frac{i}{N}}(n) \geq N \geq c2^{\frac{n}{2}}$. En changeant la valeur de c on peut obtenir la même égalité pour tout n. □

Par définition, un point grande dépendance influe sur la valeur de la norme, mais il ne la détermine pas à lui seul. Le théorème suivant indique qu'étant donnée une précision, l'ensemble de tous les points de suffisamment grande dépendance *suffit* à déterminer la valeur de la norme à cette précision.

Soit $\mathrm{I}_{n,l}$ l'ensemble des points de dépendance plus grande que l à précision n :

$$\mathrm{I}_{n,l} = \{\alpha \mid \mathrm{d}_{F,\alpha}(n) \geq l\}.$$

Remarque 3.3. *D'après la proposition 3.3, la fonction $\alpha \mapsto \mathrm{d}_{F,\alpha}(n)$ est continue, donc $I_{n,l}$ est fermé.*

Nous allons maintenant montrer qu'une fonction nulle sur un un tel ensemble a une petite norme, et pour cela nous avons besoin du lemme suivant.

Lemme 3.2. *Si $\alpha \in [0,1]$ et $g \in Lip_1$ est supportée sur $\mathcal{B}(\alpha, \frac{1}{l}) \setminus I_{n,l}$, alors $F(g) \leq 2^{-n+1}$.*

PREUVE

Si $\mathrm{d}_{F,\alpha}(n) \leq l$ alors $F(g) \leq 2^{-n}$. Dans le cas contraire, $g(\alpha) = 0$ et on peut décomposer g en une somme de deux fonctions 1-lipschitziennes g_0 et g_1 supportées resprectivement sur $]\alpha - \frac{1}{l}, \alpha[$ et $]\alpha, \alpha + \frac{1}{l}[$. Pour $i \in \{0,1\}$, soit $g_i = 0$ auquel cas $F(g_i) = 0$, ou bien il existe $\beta \in \mathrm{Supp}(g_i)$ et alors g_i est supportée sur $\mathcal{B}(\beta, \frac{1}{l})$. On a alors $\mathrm{d}_{F,\beta}(n) < l$ et donc $F(g_i) \leq 2^{-n}$. On peut alors conclure que $F(g) \leq F(g_0) + F(g_1) \leq 2^{-n+1}$. □

Théorème 3.1. *Si $l \leq \mathrm{D}_F(n)$ et $f \in Lip_1$ est nulle sur $I_{n,l}$, alors $F(f) \leq l^2 2^{-n+6}$.*

PREUVE

L'idée de la preuve est de décomposer la fonction en fonctions supportées sur de petits intervalles autour de points de faible dépendance. Leur norme est alors petite, donc de même pour celle de la fonction initiale.

On prouve tout d'abord la borne $F(f) \leq l^2 2^{-n+5}$ dans le cas où f et une fonction positive. On peut alors obtenir le cas général en décomposant f comme la différence de deux fonctions positives.

On peut supposer $l \geq 1$, car dans le cas contraire, $\mathrm{I}_{n,l} = [0,1]$ et alors $F(f) = 0$. On définit alors les ensembles $A = \{\frac{2k}{l} \mid k \in \mathbb{N}, k \leq \frac{l}{2}\}$ et $B = \{\frac{2k+1}{l} \mid k \in \mathbb{N}, k \leq \frac{l-1}{2}\}$ et les fonctions distance respectives $d_A(x) = d(x,A)$ et $d_B(x) = d(x,B)$. On remarque alors que $d_A + d_B = \frac{1}{l}$.

Étant donné f positive, on définit les fonction g_n et f_n suivantes pour tout $n \geq 0$, représentées dans la figure 3.3 :

$$g_{2n} = \min\left(\tfrac{2n}{l} + d_A, f\right), \qquad f_0 = g_0,$$
$$g_{2n+1} = \min\left(\tfrac{2n+1}{l} + d_B, f\right), \qquad f_{n+1} = g_{n+1} - g_n.$$

Pour tout $i \in \mathbb{N}$, g_i est 1-lipschitzienne, donc f_0 aussi, et f_{i+1} est 2-lipschitzienne. On a

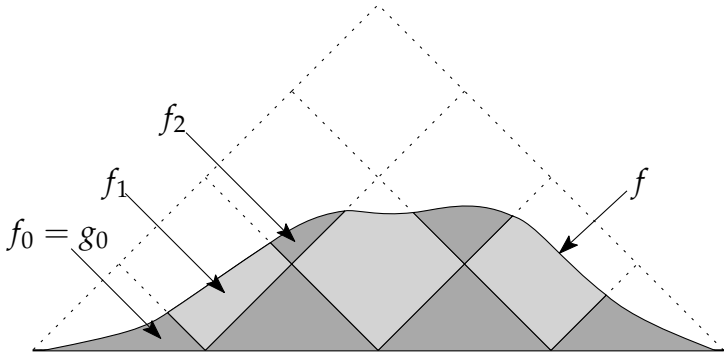

FIGURE 3.3 – Les fonctions f_i, définies par la hauteur des zones grisées.

alors $g_{2n} \leq g_{2n+1} \leq g_{2n} + 2d_B$ et $g_{2n+1} \leq g_{2n+2} \leq g_{2n+1} + 2d_A$, donc $0 \leq f_{2n+1} \leq 2d_B$ et $0 \leq f_{2n+2} \leq 2d_A$. En particulier, f_{2n+1} est nulle sur B et f_{2n} est nulle sur A.

Comme $\mathrm{Supp}(f)$ est disjoint de $\mathrm{I}_{n,l}$ et que $l \leq \mathrm{D}_F(n)$, on a $\mathrm{I}_{n,l} \neq \emptyset$ et donc f s'annule en un point. Comme de plus f est 1-lipschitzienne, $\|f\|_\infty \leq 1$ et donc $f_k = 0$ pour tout $k \geq l+1$ puisque $g_k = g_{k-1} = f$.

Pour tout i on a alors $\mathrm{Supp}(f_i) \subseteq \mathrm{Supp}(f)$ qui est aussi disjoint de $\mathrm{I}_{n,l}$. Comme f_i est non nulle sur A ou sur B, $\frac{f_i}{2}$ est la somme d'au plus $\frac{l+3}{2}$ fonctions satisfaisant les conditions du lemme 3.2, donc $F(f_i) \leq (l+3)2^{-n+1}$. Comme $f = f_0 + \ldots + f_k$ où $k = \lceil l+1 \rceil - 1 < l+1$, on a finalement $F(f) \leq (l+2)(l+3)2^{-n+1} \leq l^2 2^{-n+5}$ étant donné que $l \geq 1$. □

Ce résultat montre que l'on peut définir une stratégie d'évaluation de la norme en connaissant les ensembles $\mathrm{I}_{n,l}$. En effet, si $\varepsilon > 0$ et $l = 2^{\frac{n-7}{2}}\sqrt{\varepsilon}$, le théorème 3.1 implique que si f et g sont des fonctions de Lip_1 qui coïncident sur $\mathrm{I}_{n,l}$, alors $|F(f) - F(g)| \leq F(f-g) \leq \varepsilon$ (en l'appliquant à $\frac{f-g}{2} \in Lip_1$). Pour connaître $F(f)$ à ε près, il suffit alors de connaître f sur $\mathrm{I}_{n,l}$.

Points importants

L'intuition est qu'une norme dépend de la valeur de son argument, principalement aux points de grande dépendance. Il est alors naturel de se demander si l'on peut déterminer les points de $[0,1]$ qui sont nécessairement évalués par une machine qui calcule cette norme. Pour cela nous définissons un ensemble de points dits *importants* en fixant une borne inférieure sur la croissance asymptotique de la dépendance.

L'exemple de la norme L^1 montre que la croissance de toute dépendance est d'au moins $n \mapsto 2^{\frac{n}{2}}$, c'est donc une borne inférieure sur le seuil que nous souhaitons choisir.

Comme suggéré dans la proposition 3.5, il semble (et c'est effectivement le cas comme nous le montrerons dans la proposition 3.6) qu'il existe toujours des points dont la dépendance est de l'ordre de $n \mapsto 2^{\frac{n}{2}}$. Ils forment donc un bon candidat pour définir l'ensemble de points importants décrit plus haut.

On définit alors l'ensemble des points importants relativement à une norme à partir de cette borne. Ce choix sera alors justifié par les théorèmes 3.2, 3.3 et 3.4.

Définition 3.6 (Points importants)**.** *Si F est une norme sur $\mathcal{C}[0,1]$, alors un point $\alpha \in [0,1]$ est* **important** *relativement à F s'il existe $c > 0$ tel que pour tout entier n, $\mathrm{d}_{F,\alpha}(n) \geq c \cdot 2^{\frac{n}{2}}$.*

Lemme 3.3. *L'ensemble des points importants est une union croissante dénombrable d'ensembles compacts.*

Preuve

Par définition des points importants on a $\mathcal{I} = \bigcup_k \mathcal{I}_k$ avec $\mathcal{I}_k = \{\alpha \mid \forall n, \mathrm{d}_{F,\alpha}(n) \geq 2^{\frac{n}{2}-k}\} = \bigcap_n \mathrm{I}_{n,2^{\frac{n}{2}-k}}$. On a $\mathcal{I}_k \subseteq \mathcal{I}_{k+1}$ et d'après la remarque 3.3 $\mathcal{I}_k$ est fermé pour tout k, ce qui donne le résultat. □

L'ensemble des points importants est en effet non vide et est même dense. C'est en effet nécessaire car une norme doit intuitivement dépendre de son argument sur tout l'intervalle $[0,1]$ (autrement le premier point de la définition 3.2 ne serait pas vérifié).

Proposition 3.6. *L'ensemble des points importants de toute norme est dense dans $[0,1]$.*

Preuve

Soit $p \in \mathbb{N} \setminus \{0\}$ et $a \in [2^{-p}, 1-2^{-p}]$. On construit un point important $\alpha \in \overline{\mathcal{V}}(a, 2^{-p})$. Soit $c \in \mathbb{N}$ tel que $F(h_{a,2^{-p}}) > 2^{-c}$. α sera la limite d'une suite $(\alpha_n)_{n\in\mathbb{N}}$, définie par induction sur n, vérifiant $F(h_{\alpha_n,2^{-n-p}}) > 2^{-2n-c}$.

On commence avec $\alpha_0 = a$, puis étant donné α_n, on décompose $h_{\alpha_n,2^{-n-p}}$ en une somme de quatre fonctions (c.f. figure 3.4). On pose $\beta_1 = \beta_2 = \alpha_n$, $\beta_3 = \alpha_n - 2^{-n-p-1}$ et $\beta_4 =$

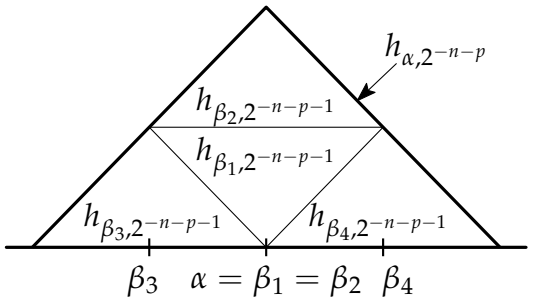

Figure 3.4 – Représentation graphique de la décomposition de $h_{\alpha_n,2^{-n-p}}$ en $\sum_{i=1}^{4} h_{\beta_i,2^{-n-p-1}}$.

$\alpha_n + 2^{-n-p-1}$. On a alors $h_{\alpha_n,2^{-n-p}} = \sum_{i=1}^{4} h_{\beta_i,2^{-n-p-1}}$ donc par l'inégalité triangulaire, il existe un point α_{n+1} parmi $\{\beta_1, \beta_2, \beta_3, \beta_4\}$ tel que $F(h_{\alpha_{n+1},2^{-n-p-1}}) \geq F(h_{\alpha_n,2^{-n-p}})/4 > 2^{-2n-2-c}$.

Comme $|\alpha_n - \alpha_{n+1}| \leq 2^{-n-p-1}$, $(\alpha_n)_{n\in\mathbb{N}}$ converge et on nomme α sa limite. On a alors $|\alpha - \alpha_n| \leq 2^{-n-p}$ et donc $h_{\alpha_n,2^{-n-p}}$ est supportée sur $\mathcal{B}(\alpha, 2^{-n-p+1})$. Comme de plus $F(h_{\alpha_n,2^{-n-p}}) > 2^{-2n-c}$, $\mathrm{d}_{F,\alpha}(2n+c) \geq 2^{n+p-1}$. Enfin, on choisit $k \geq \frac{c+3}{2} - p$ et alors pour tout $n > 2k > c$, $\mathrm{d}_{F,\alpha}(n) \geq 2^{\frac{n}{2}-k}$. □

Remarque 3.4. *La proposition 3.4 implique que si F est une norme plus faible que G, alors tout point important pour F est important pour G.*

Exemple 3.6. *Pour la norme uniforme, tout point est important. En particulier, $\mathcal{I} = \mathcal{I}_0 = [0,1]$. En effet, intuitivement, et en observant sa définition, la norme uniforme dépend de la même manière de tous les points de $[0,1]$, ils sont donc tous importants.*

Exemple 3.7. *De même, pour la norme L^1, tout point est important, et $\mathcal{I} = \mathcal{I}_1 = [0,1]$.*

Exemple 3.8. *Soit F_q une norme telle que définie dans l'exemple 3.2. L'exemple 3.5 montre que tous les q_i sont importants. Selon la façon dont sont ces points sont distribués sur $[0,1]$, ils se peut ou non qu'ils soient les seuls points importants.*

1. *Soit $(q_i)_{i\in\mathbb{N}}$ l'énumération usuelle des nombres rationnels dyadiques. Autrement dit, si $i = 2^n + k$ avec $k < 2^n$, alors $q_i = (2k+1)2^{-n}$. Cette énumération répartit les dyadiques de façon à ce qu'ils soient uniformément espacés les uns des autres. Plus précisément, si $i < j$, alors $|q_i - q_j| \geq \frac{1}{j}$.*

   Lemme 3.4. *Si $(q_i)_{i\in\mathbb{N}}$ est l'énumération usuelle des dyadiques, alors $\mathcal{I} = \mathbb{D}$. De plus, si $\alpha \notin \mathbb{D}$, alors $\mathrm{d}_{F_q,\alpha}(n) \leq 2(n+1)$ pour une infinité de n.*

   Preuve

   Soit $\alpha \notin \mathbb{D}$ et $n_0 \in \mathbb{N}$. Supposons que pour tout $n \geq n_0$, $\mathrm{d}_{F_q,\alpha}(n) > 2(n+1)$.
   Comme F_q est une norme monotone, on a, d'après 3.2, $F_q(h_{\alpha,\frac{1}{2(n_0+1)}}) > 2^{-n_0}$ donc il existe $i \leq n_0$ tel que $|\alpha - q_i| < \frac{1}{2(n_0+1)}$ (dans le cas contraire, les n_0+1 premiers termes de la somme définissant F_q seraient nuls et la norme serait bornée par 2^{-n_0}). On choisit $i \leq n_0$ minimisant $|\alpha - q_i|$. Comme $\alpha \neq q_i$ il existe $n > n_0$ minimal tel que $|\alpha - q_i| \geq \frac{1}{2(n+1)}$. On a toujours l'inégalité $F_q(h_{\alpha,\frac{1}{2(n+1)}}) > 2^{-n}$, donc il existe $j \leq n$ tel que $|\alpha - q_j| < \frac{1}{2(n+1)}$. On a donc $|\alpha - q_j| < |\alpha - q_i|$ avec $j > n_0 \geq i$. Enfin, $|q_i - q_j| \leq |\alpha - q_i| + |\alpha - q_j| < \frac{1}{2n} + \frac{1}{2(n+1)} < \frac{1}{n} \leq \frac{1}{j}$ ce qui fournit une contradiction.
   On a donc $\mathrm{d}_{F_q,\alpha}(n) \leq 2(n+1)$ pour une infinité de n, et en particulier α n'est pas un point important. □

2. *Inversement, si les $(q_i)_{i\in\mathbb{N}}$ ne sont pas répartis uniformément, par exemple s'ils s'accumulent rapidement autour d'un point $\alpha \notin \mathbb{D}$, alors ce point peut être important. Par exemple, si $|q_{2i} - \alpha| < 2^{-2i}$, alors on peut montrer que pour tout n, $\mathrm{d}_{F_q,\alpha}(n) \geq 2^{\frac{n}{2}-2}$, et donc α est un point important.*

Le théorème suivant justifie la terminilogie de *points importants* en montrant que la valeur de $F(f)$ à une précision donnée dépend seulement de la valeur de f sur $\mathcal{I}_k$.

Théorème 3.2. *Soit F est une norme plus faible que la norme uniforme. Il existe une constante c telle que si $f \in Lip_1$ est nulle sur $\mathcal{I}_{2k}$, alors $F(f) \leq c \cdot 2^{-k}$.*

Pour prouver ce résultat nous avons besoin de quelques lemmes techniques.

Lemme 3.5. *Pour tout $\alpha \in [0,1]$ et $n \in \mathbb{N}$ il existe $\beta \in [0,1]$ tel que $|\alpha - \beta| \leq 1/(2\mathrm{d}_{F,\alpha}(n))$ et $\mathrm{d}_{F,\beta}(n+2) \geq 2\mathrm{d}_{F,\alpha}(n)$.*

Preuve

Soit $l < \mathrm{d}_{F,\alpha}(n)$ et $f \in Lip_1$ supportée sur $\mathcal{B}(\alpha, \frac{1}{l})$ telle que $F(f) > 2^{-n}$. Soient h_1 et h_2 les fonctions 1-lipschitziennes maximales supportées sur $[0,\alpha]$ et $[\alpha,1]$ respectivement. On pose alors $f_1 = \max(\min(f,h_1),-h_1)$, $f_2 = \max(\min(f,h_2),-h_2)$ et $f_3 = f_4 = (f - f_1 - f_2)/2$. Ces fonctions sont 1-lipschitziennes et il existe $\beta_i \in \overline{\mathcal{V}}(\alpha, \frac{1}{2l})$ tel qu'elles sont supportées

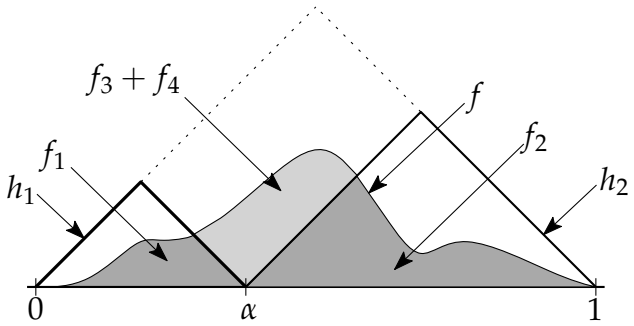

Figure 3.5 – Décomposition de f en $f_1 + f_2 + f_3 + f_4$.

respectivement sur $\mathcal{B}(\beta_i, \frac{1}{2l})$. Comme $f = f_1 + f_2 + f_3 + f_4$, par l'inégalité triangulaire, il existe $1 \leq i \leq 4$ tel que $F(f_i) > 2^{-n-2}$, ce qui implique que $d_{\beta_i}(n+2) > 2l$.

On a donc associé à chaque $l < \mathrm{d}_{F,\alpha}(n)$ un point β_l. Par compacité de l'intervalle $[0,1]$il existe un point d'accumulation β lorsque l tend vers $\mathrm{d}_{F,\alpha}(n)$. Par continuité, β vérifie les conditions du lemme. □

Lemme 3.6. *Pour tout $\alpha \in [0,1]$ et tout $n \in \mathbb{N}$ il existe $\beta \in [0,1]$ tel que $|\alpha - \beta| \leq \frac{1}{\mathrm{d}_{F,\alpha}(n)}$ et pour tout $p \in \mathbb{N}$, $\mathrm{d}_{F,\beta}(n+p) \geq 2^{\frac{p-3}{2}}\mathrm{d}_{F,\alpha}(n)$.*

Preuve

On applique itérativement le lemme 3.5 : on pose $\beta_0 = \alpha$, et étant donné β_p, on obtient β_{p+1} en appliquant le lemme 3.5 à β_p et $n+2p$. On a alors $\mathrm{d}_{F,\beta_p}(n+2p) \geq 2^p\mathrm{d}_{F,\alpha}(n)$ et $|\beta_p - \beta_{p+1}| \leq 2^{-p-1}/\mathrm{d}_{F,\alpha}(n)$. La suite $(\beta_n)_{n\in\mathbb{N}}$ est alors une suite de Cauchy et on appelle sa limite β. Comme $|\beta - \beta_p| \leq 2^{-p}/\mathrm{d}_{F,\alpha}(n)$, toute fonction supportée sur $\mathcal{B}(\beta_p, 2^{-p}/\mathrm{d}_{F,\alpha}(n))$ est supportée sur $\mathcal{B}(\beta, 2^{-p+1}/\mathrm{d}_{F,\alpha}(n))$, donc $\mathrm{d}_{F,\beta}(n+2p) \geq 2^{p-1}\mathrm{d}_{F,\alpha}(n)$. Ainsi, comme $\mathrm{d}_{F,\beta}(n+2p+1) \geq \mathrm{d}_{F,\beta}(n+2p) \geq 2^{p-1}\mathrm{d}_{F,\alpha}(n)$ et on obtient le résultat. □

Le lemme suivant est une version plus précise du théorème 3.1.

Lemme 3.7. *Il existe une constante c telle que pour tout $\varepsilon > 0$ et tout $f \in Lip_1$, si $|f| \leq \varepsilon$ sur $I_{n,l}$, alors $F(f) \leq c\varepsilon + l^2 2^{-n+6}$.*

Preuve

On décompose $f \in Lip_1$ en une somme $f = g + h$ de fonctions 1-lipschitziennes telles que $\|g\|_\infty \leq \varepsilon$ et $h = 0$ sur $\mathrm{I}_{n,l}$. Les fonctions g et h sont définies par $g(x) = \max(\min(f(x), \varepsilon), -\varepsilon)$ et $h = f - g$. On vérifie facilement que g et h sont 1-lipschitziennes. Si $|f| \leq \varepsilon$ sur $\mathrm{I}_{n,l}$, sur cet ensemble, $f = g$, et donc $h = 0$. Enfin, par l'inégalité triangulaire, $F(f) \leq F(g) + F(h)$. Comme $F(g) \leq c_F\varepsilon$ et $F(h) \leq l^2 2^{-n+6}$ d'après le théorème 3.1, on obtient le résultat souhaité. □

Preuve

D'après le lemme 3.6,

$$\mathrm{I}_{2k,l} \subseteq \mathcal{B}\left(\bigcap_{n\geq 2k} \mathrm{I}_{n,2^{\frac{n-3}{2}-k}l'}, \frac{1}{l}\right).$$

Pour tout l plus grand que $2^{\frac{3}{2}}$, l'ensemble $\bigcap_{n\geq 2k} \mathrm{I}_{n,2^{\frac{n-3}{2}-k}l}$ est inclus dans $\mathcal{I}_k$. En effet, si $n \geq 2k$, alors $l2^{\frac{n-3}{2}-k} \geq 2^{\frac{n}{2}-k}$.

Une fonction $f \in Lip_1$ nulle sur $\mathcal{I}_k$ vérifie donc $|f| \leq \frac{1}{l}$ sur $\mathrm{I}_{2k,l}$, donc d'après le lemme 3.7 il existe une constante c telle que $F(f) \leq \frac{c}{l} + l^2 2^{-2k+6}$. Donc avec $k \geq 3$ et $l = 2^{\frac{k}{2}}$ on a : $F(f) \leq c2^{-\frac{k}{2}} + 2^{-k+6} \leq c'2^{-\frac{k}{2}}$ pour une certaine constante c' qui ne dépend que de c. En choisissant une nouvelle constante pour couvrir les cas $k < 3$ on obtient le résultat. □

On peut maintenant généraliser ce résultat à toute fonction $f \in \mathcal{C}[0,1]$.

Corollaire 3.1. *Si F est une norme plus faible que la norme uniforme, alors il existe une constante $c \in \mathbb{N}$ telle que si $f \in \mathcal{C}[0,1]$ est nulle sur $\mathcal{I}_{2\mu_f(p)}$ (où μ_f est un module de continuité pour f), alors $F(f) \leq c \cdot 2^{-p}$.*

Preuve

Soient $L = 2^{\mu_f(p)-p+1}$ et $k = \mu_f(p)$. Il existe une fonction L-lipschitzienne g nulle sur $\mathcal{I}_{2k}$ telle que $\|f - g\|_\infty \leq 2^{-p}$. En effet, soit $p_i = i2^{-\mu_f(p)}$ pour $0 \leq i \leq 2^{\mu_f(p)}$. Soit $g(p_i) = 0$ si $d(p_i, \mathcal{I}_{2k}) < 2^{-\mu_f(p)}$, $g(p_i) = f(p_i)$, dans le cas contraire on étend g par une fonction linéaire entre deux points. On peut vérifier facilement que g vérifie les conditions demandées.

Comme $F(f - g) \leq c_F 2^{-p}$, en applicant le théorème 3.2 à $\frac{g}{L}$ (qui est une fonction 1-lipschitzienne) on obtient $F(g) \leq cL2^{-\mu_f(p)} \leq c_1 2^{-p}$pour une certaine constante c, et donc $F(f) \leq c_F 2^{-p} + c2^{-p}$. □

3.1.3 Complexité déterministe

Nous allons maintenant faire le lien entre la complexité d'une norme et ses propriétés analytiques, en particulier avec la taille de son ensemble de points importants. En effet, restreindre la complexité d'une machine à oracle a deux influences principales, la première étant que sa puissance de calcul est réduite (au sens où elle ne peut

simuler des problèmes d'ordre 1 intrinsèquement difficiles). Nous nous intéresseront seulement à la seconde qui est due à la limitation du nombre d'appels à l'oracle. En effet, borner la complexité d'appel borne la quantité d'information que la norme peut obtenir de son entrée, et en particulier le nombre de points dont elle peut dépendre.

Pour cela, nous allons utiliser la notion de *complexité d'appel* établie dans la définition 1.10.

Étant donné les propriétés des normes nous pouvons nous contenter d'étudier leurs complexité d'appel seulement sur la fonction nulle. On utilisera donc une représentation z de la fonction nulle (pour $\delta_\square$) dont le module de continuité est $\mu_z(p) = p$ et la fonction d'approximation est définie par $z_\mathbb{Q}(q, p) = 0$. On notera de plus que la taille de cette représentation est de l'ordre de la fonction identité. On dira alors (par extension de la définition 1.10) qu'une norme $\|\cdot\|$ a une complexité d'appel $t : \mathbb{N} \to \mathbb{N}$ si elle est calculée par une machine à oracle dont la complexité d'appel sur z et la précision n est bornée par $t(n)$. On notera alors Q_n l'ensemble (de taille inférieure à $t(n)$) des nombres rationnels fournis à l'oracle pour évaluer la fonction d'approximation de z.

Dans la suite, nous étudierons plus spécifiquement les propriétés des normes dont la complexité d'appel est sous-exponentielle. En particulier, étant donné qu'une borne sur la complexité est aussi une borne sur la complexité d'appel, une norme calculable en temps $T : (\mathbb{N} \to \mathbb{N}) \to \mathbb{N} \to \mathbb{N}$ a une complexité d'appel bornée par $n \mapsto T(|z|_1, n) \leq n \mapsto T(id, n)$, donc toute norme calculable en temps polynomial (d'ordre 2) a une complexité d'appel bornée par un polynôme (d'ordre 1). On notera de plus que toute norme de complexité sous-exponentielle est plus faible que la norme uniforme, d'après la proposition 3.2, et la plupart des résultats précédents s'appliquent donc.

L'intuition qui permet de faire le lien entre l'ensemble des points importants et la complexité d'appel est la suivante : si une norme dépend d'un point, alors une machine qui la calcule doit nécessairement faire des appels à sa fonction d'approximation autour de ce point.

On considère maintenant une norme F calculée par une machine $\mathcal{M}$, Q_n son ensemble d'appels sur z et n et $\mathcal{T}$ une classe de fonctions de $\mathbb{N} \to \mathbb{N}$ close par composition avec des polynômes et fermée par le bas (*i.e.* si $t \leq t'$ alors $t' \in \mathcal{T}$ implique $t \in \mathcal{T}$).

Lemme 3.8. *Pour tout $\alpha \in [0, 1], n \in \mathbb{N}$ on a :*

$$I_{n,l} \subseteq \overline{\mathcal{V}}(Q_{n+1}, \frac{1}{l})$$
$$\alpha \in \overline{\mathcal{V}}(Q_{n+1}, \frac{1}{\mathrm{d}_{F,\alpha}(n)})$$
$$\mathrm{d}_{F,\alpha}(n) \leq \frac{1}{d(\alpha, Q_{n+1})}.$$

Preuve

Ces trois proposition sont équivalentes, il suffit de prouver la troisième. On pose $l = \frac{1}{d(\alpha, Q_{n+1})}$. Si $f \in Lip_1$ est supportée sur $\mathcal{B}(\alpha, \frac{1}{l})$, alors f est nulle sur Q_{n+1}. On peut alors choisir une représentation de f telle que $f_{\mathbb{Q}}(q,p) = 0$ si $q \in Q_{n+1}$. La machine $\mathcal{M}$ ne peut distinguer cette représentation de z à précision $n+1$, elle doit donc produire le même rationnel r tel que $|r| = |r - F(0)| < 2^{-(n+1)}$. Comme de plus $|r - F(f)| < 2^{-(n+1)}$, a alors $F(f) < 2^{-n}$. Par définition de la dépendance, on a donc $\mathrm{d}_{F,\alpha}(n) \leq l$. □

Théorème 3.3. *Si F a une complexité d'appel dans $\mathcal{T}$, alors pour presque tout $\alpha \in [0,1]$, $\mathrm{d}_{F,\alpha} \in \mathcal{T}$.*

En particulier, si $\mathcal{T}$ est une classe sous-exponentielle, alors l'ensemble des points importants de F est de mesure nulle.

Preuve

Soit $t \in \mathcal{T}$ tel que pour tout $n \in \mathbb{N}$, $|Q_n| \leq t(n)$. On pose $U_i = \bigcup_n \mathcal{B}(Q_n, \frac{1}{2t(n)(n+i+1)^2})$ et $A_i = [0,1] \setminus U_i$. On a alors

$$\lambda_{[0,1]}(U_i) \leq \sum_n \frac{1}{(n+i+1)^2} \leq \frac{1}{i}$$

où $\lambda_{[0,1]}$ désigne la mesure de Lebesgue sur $[0,1]$.

On a alors $\lambda_{[0,1]}(A_i) \geq 1 - \frac{1}{i}$. Soient $\alpha \in A_i$ et $n \in \mathbb{N}$. D'après le lemme 3.8, on a $\alpha \notin \mathcal{B}(Q_n, \frac{1}{2t(n)(n+i+1)^2})$, $\mathrm{d}_{F,\alpha}(n) \leq 1/d(\alpha, Q_{n+1}) \leq 2t(n+1)(n+i+2)^2$ et la fonction $n \mapsto 2t(n+1)(n+i+2)^2$ appartient bien à $\mathcal{T}$ car $\mathcal{T}$ est stable par composition avec des polynômes.

Le second point est une implication directe du premier, par définition de la dépendance. □

On a même le résultat plus fort suivant :

Théorème 3.4. *Si F a une complexité d'appel dans une classe sous-exponentielle $\mathcal{T}$, alors l'ensemble des points importants de F a une dimension de Hausdorff nulle.*

Preuve

Dans la preuve précédente, étant donné $s > 0$, on remplace les ensembles U_i par $V_i^s = \bigcup_n \mathcal{B}\left(Q_n, (t(n)(n+i)^2)^{-\frac{1}{s}}\right)$.

$$V_i^s = \bigcup_n \mathcal{B}\left(Q_n, \frac{1}{(t(n)(n+i)^2)^{\frac{1}{s}}}\right).$$

De même, $\bigcap_i V_i^s$, dont la dimension est inférieure à s, contient l'ensemble des points importants, étant donné que $(t(n)(n+i)^2)^{\frac{1}{s}} = o(2^{\frac{n}{2}})$. Comme c'est vrai pour tout $s > 0$, l'ensemble des points importants est de dimension nulle. □

Remarque 3.5. *Ces théorèmes permettent d'obtenir une nouvelle preuve de la proposition 3.1 étant donné que pour la norme uniforme et la norme L^1 l'ensemble des points importants est $[0,1]$.*

Si F est une norme calculable en temps polynomial, alors les théorèmes précédents s'appliquent et la plupart des points ne sont pas importants. Autrement dit, F dépend d'un ensemble «très petit» de points. Nous allons maintenant montrer que cela est contrebalancé par le fait que la dépendance en certains points est au contraire très grande.

Selon la proposition 3.6 il existe un ensemble dense de points dont la dépendance est au moins de l'ordre de $2^{\frac{n}{2}}$, et l'exemple de la norme L^1 montre que le coefficient $\frac{1}{2}$ est maximal dans le cas général. Cependant, dans le cas des normes polynomiales, il peut être pris arbitrairement proche de 1. Pour prouver cela, commençons par améliorer la proposition 3.5.

Proposition 3.7. *Si F une norme dont la complexité d'appel est dans la classe $\mathcal{T}$, alors $n \mapsto \frac{2^n}{\mathrm{D}_F(n)} \in \mathcal{T}$.*

Preuve

Soit $t \in \mathcal{T}$ tel que pour tout $\in \mathbb{N}$, $|Q_n| \leq t(n)$. Soient $l < F(1)2^n/(3t(n))$ et $g = \frac{1}{t(n)} \sum_{q \in Q_n} h_{q,\frac{1}{l}}$. On a $g \in Lip_1$ et $g = x \mapsto 1/(lt(n))$ sur Q_n ce qui implique que $F(1/(lt(n)) - g) \leq 2^{-n+1}$ et donc $F(g) \geq F(1)/(lt(n)) - 2^{-n+1}$. Il existe alors $q \in Q_n$ tel que $F(h_{q,1/l}) \geq F(1)/(lt(n)) - 2^{-n+1} > 2^{-n}$, donc $\mathrm{D}_F(n) \geq \mathrm{d}_{F,q}(n) \geq l$. En maximisant sur $l < F(1)2^n/(3t(n))$, on obtient $\mathrm{D}_F(n) \geq F(1)2^n/(3t(n))$. □

Cela s'applique en particulier aux normes de complexité d'appel polynomiale.

Corollaire 3.2. *Pour tout norme F donc la complexité d'appel est polynomiale, il existe un polynôme P tel que la dépendance maximale de F vérifie $D_F(n) \geq \frac{2^n}{P(n)}$ pour tout $n \in \mathbb{N}$.*

Remarque 3.6. *On peut appliquer ce résultat pour prouver d'une autre manière qu'une norme polynomiale ne peut être plus faible que la norme L^1 (proposition 3.2). En effet, la dépendance maximale de la norme L^1 est de l'ordre de $2^{\frac{n}{2}}$, ce qui contredirait la proposition 3.4.*

On peut maintenant se demander si la borne donnée par la proposition 3.7 est atteinte, *i.e.* existe-t-il au moins un point $\alpha \in [0,1]$ tel que $n \mapsto \frac{2^n}{\mathrm{d}_{F,\alpha}(n)} \in \mathcal{T}$?

Dans le cas général la question reste ouverte, mais dans le cas où $\mathcal{T}$ est la classe des polynômes, alors on peut montrer que pour tout $\varepsilon > 0$, on peut trouver α, tel que $\mathrm{d}_{F,\alpha}$ est plus grand que $2^{(1-\varepsilon)n}$.

Théorème 3.5. *Soit F une norme calculable en temps polynomial. Il existe $\alpha \in [0,1]$ et $c > 0$ tels que $\mathrm{d}_{F,\alpha}(n) \geq 2^{n - c\sqrt{n}\log n}$ pour tout n suffisamment grand. De plus, l'ensemble de ces points est dense dans [0,1].*

Preuve

L'idée est de partir d'une fonction triangulaire $h_{\alpha_0,l}$, puis de la décomposer en somme de fonctions triangulaires plus petites. Étant donné que la plupart d'entre elles seront loin de l'ensemble des requêtes effectuées par la machine qui calcule la norme, leur norme sera très petite.

Par l'inégalité triangulaire, l'une des autres fonctions (proches de l'ensemble des appels) doit avoir une grande norme. On itère alors pour obtenir une suite $(\alpha_n)_{n\in\mathbb{N}}$ qui va converger vers une limite α vérifiant les hypothèses du théorème.

Nous avons tout d'abord besoin du lemme suivant

Lemme 3.9. *Soient $k, p, n \in \mathbb{N}$ et $\alpha \in [2^{-k}, 1-2^{-k}]$. Il existe alors $\beta \in \mathcal{B}(\alpha, 2^{-k})$ tel que*

$$F(h_{\beta,2^{-p-k}}) \geq \frac{1}{P(n)}(2^{-p-1}F(h_{\alpha,2^{-k}}) - 2^{p-n+1}),$$

où P borne la complexité d'appel de F.

Preuve

On décompose $h_{\alpha,2^{-k}}$ en une somme de 2^{2p} fonctions $h_{\beta_i,2^{-p-k}}$, $i = 1\ldots, 2^{2p}$ où les β_i décrivent l'ensemble $B = \{\alpha \pm j2^{-p-k} : 0 \leq j < 2^p\}$ de taille $2^{p+1}-1$. Chaque β_i est le centre du support d'au plus 2^p fonctions. On a donc

$$F(h_{\alpha,2^{-k}}) \leq 2^p \sum_{\beta\in B} F(h_{\beta,2^{-p-k}}).$$

On partitionne B en $B_0 \cup B_1$ ainsi : $\beta \in B_0$ si et seulement si $d(\beta, Q_n) < 2^{-p-k}$. Étant donné que l'on peut associer à chaque $\beta \in B_0$ un point q dans Q_n tel que $|\beta - q| < 2^{-p-k}$ et qu'au plus deux tels β peuvent être associés à un tel q, on a $|B_0| \leq 2|Q_n|$.

On pose $M = \max_{\beta\in B_0} F(h_\beta, 2^{-p-k})$. On a alors $M = 0$ si $B_0 = \emptyset$. Si $\beta \in B_1$ alors $F(h_{\beta,2^{-p-k}}) \leq 2^{-n+1}$ et donc

$$\begin{aligned} F(h_{\alpha,2^{-k}}) &\leq 2^p \sum_{\beta\in B_0} F(h_{\beta,2^{-p-k}}) + 2^p \sum_{\beta\in B_1} F(h_{\beta,2^{-p-k}}) \\ &\leq 2^p|B_0|M + 2^{p-n+1}|B_1| \\ &\leq 2^{p+1}P(n)M + 2^{2p-n+2}. \end{aligned}$$

On a donc :

$$M \geq \frac{1}{P(n)}(2^{-p-1}F(h_{\alpha,2^{-k}}) - 2^{p-n+1}).$$

Si le terme de droite est strictement positif, alors M aussi et donc B_0 est non vide. Il existe alors $\beta \in B_0$ tel que $M = F(h_{\beta,2^{-p-k}})$. Dans le cas contraire, tout β vérifie alors les conditions du lemme. □

On choisit maintenant k_0 et d des entiers vérifiant pour tout $k \geq k_0$,

$$\frac{2^{-p-2-k-\sqrt{k}}}{P(2p+4+k+\sqrt{k})} > 2^{-(p+k)-\sqrt{p+k}}$$

où $p = \lceil d\sqrt{k}\log(k)\rceil$.

On peut en effet vérifier qu'en prenant $d = 2d_0 + 1$, où d_0 est le degré de P, le rapport entre les deux membres de l'équation diverge vers $+\infty$ et est alors plus grand que 1 à partir d'un certain rang k_0.

Lemme 3.10. *Pour tout $k \geq k_0$, si $F(h_{\alpha,2^{-k}}) > 2^{-k-\sqrt{k}}$ alors il existe un point $\beta \in \mathcal{B}(\alpha, 2^{-k})$ tel que $F(h_{\beta,2^{-p-k}}) > 2^{-p-k-\sqrt{p+k}}$ où $p = \lceil d\sqrt{k}\log k \rceil$.*

Preuve

On applique le lemme 3.9 à $p = \lceil d\sqrt{k}\log k \rceil$ et $n = \lceil 2p + 3 + k + \sqrt{k} \rceil$. Obtient alors $\beta \in \mathcal{B}(\alpha, 2^{-k})$ vérifiant

$$F(h_{\beta,2^{-p-k}}) \geq \frac{2^{-p-2-k-\sqrt{k}}}{P(2p+4+k+\sqrt{k})}.$$

On a choisi k et d de façon à ce que le terme de droite soit plus grand que $2^{-(p+k)-\sqrt{p+k}}$, ce qui fournit bien le résultat. □

On définit enfin les suites $(\alpha_n)_{n\in\mathbb{N}}, (p_n)_{n\in\mathbb{N}}, (k_n)_{n\in\mathbb{N}}$ par induction. On commence avec k_0, $\alpha_0 \in [2^{-k_0}, 1-2^{-k_0}]$ et on suppose que $F(h_{\alpha_0,2^{-k_0}}) > 2^{-\lambda k_0}$. si ce n'est pas le cas, on multiplie F par une constante. On notera que si le résultat est valable pour cette nouvelle norme, alors il est aussi valable pour F.

À l'étape i, le lemme 3.10 appliqué à α_i et k_i fournit un point β. On pose alors $p_i = \lceil d\sqrt{k_i}\log(k_i) \rceil$, $\alpha_{i+1} = \beta$ et $k_{i+1} = k_i + p_i$. On sait alors que $F(h_{\alpha_i,2^{-k_i}} > 2^{-k_i-\sqrt{k_i}}$ et $|\alpha_i - \alpha_{i+1}| < 2^{-k_i}$ pour tout i.

Comme $(k_n)_{n\in\mathbb{N}}$ est strictement croissant, la suite $(\alpha_n)_{n\in\mathbb{N}}$ converge vers une limite α vérifiant $|\alpha_i - \alpha| \leq 2^{-k_i+1}$ pour tout $i \in \mathbb{N}$.

On pose, pour tout $i \in \mathbb{N}$, $n_i = \lceil k_i + \sqrt{k_i} \rceil$. La fonction $h_{\alpha_i,2^{-k_i}}$ est alors supportée sur $\mathcal{B}(\alpha, 2^{-k_i+2})$ et $F(h_{\alpha_i,2^{-k_i}}) > 2^{-k_i-\sqrt{k_i}} \geq 2^{-n_i}$ et donc $\mathrm{d}_{F,\alpha}(n_i) \geq 2^{k_i-2}$.

Comme $n_{i+1} - k_i \sim k_{i+1} - k_i \sim d\sqrt{k_i}\log k_i$, pour i suffisamment grand on a $n_{i+1} - k_i \leq (d+1)\sqrt{k_i}\log k_i \leq (d+1)\sqrt{n_i}\log(n_i)$.

Étant donné $n \in \mathbb{N}$ et i suffisamment grand vérifiant $n_i \leq n < n_{i+1}$, on a $k_i \geq n - (d+1)\sqrt{n}\log n$, donc $\mathrm{d}_{F,\alpha}(n) \geq \mathrm{d}_{F,\alpha}(n_i) \geq 2^{k_i-2} \geq 2^{n-(d+1)\sqrt{n}\log n-2}$.

On a donc finalement pour tout n suffisamment grand, $\mathrm{d}_{F,\alpha}(n) \geq 2^{n-(d+2)\sqrt{n}\log n}$. □

Nous ne savons pas si ce résultat peut être amélioré, autrement dit si la borne définie dans la proposition 3.7 peut être atteinte. L'adaptation de la preuve du théorème 3.5 nous permet seulement de conclure à l'existence d'un α tel que $\mathrm{d}_{F,\alpha}(n) \geq \frac{2^n}{P(n)}$ pour une infinité de $n \in \mathbb{N}$.

Nous pouvons maintenant caractériser les normes qui ont une complexité d'appel polynomiale.

Définition 3.7. *Une famille de compacts $(K_n)_{n\in\mathbb{N}}$ de l'intervalle $[0,1]$ peut être couverte polynomialement s'il existe un polynôme P tel que pour tous $n, k \in \mathbb{N}$, il existe un ensemble $A_{n,k}$ d'au plus $P(n,k)$ points de $[0,1]$ tel que $K_k \subseteq \mathcal{B}(A_{n,k}, 2^{-n})$. Autrement dit, K_k peut être couvert par un ensemble polynomial de boules de taille 2^{-n}.*

Théorème 3.6. *Une norme a une complexité d'appel polynomiale si et seulement si la suite $\mathcal{I}_k$ peut être couverte polynomialement (uniformément en k).*

Preuve

Comme la norme a une complexité d'appel polynomiale, d'après le lemme 3.8, $\mathcal{I}_k \subseteq \mathrm{I}_{2(n+k),2^n} \subseteq \overline{\mathcal{V}}(Q_{2(n+k)+1}, 2^{-n})$. Comme de plus le cardinal de $Q_{2(n+k)+1}$ est polynomial en n et k, on obtient la première implication.

Inversement, supposons $\mathcal{I}_k \subseteq \mathcal{B}(A_{n,k}, 2^{-n})$ pour un ensemble $A_{n,k}$ dont le cardinal est borné par $P(n,k)$ où P est un polynôme. On peut supposer sans perte de généralité que les points de $A_{n,k}$ sont dans l'ensemble $\mathbb{D}_n$ des dyadiques de taille n. On construit l'oracle additionnel pour qu'il fournisse deux types d'informations : étant donné n, k, il fournit l'ensemble $A_{n,k}$, et étant donné n, k, p et une liste de valeurs pour chaque points de $A_{n,k}$, il fournit une approximation à 2^{-p} près de la norme sur la fonction linéaire par morceaux f définie par ces valeurs. On peut facilement vérifier que ces informations peuvent être encodées dans un oracle de type $\mathbb{N} \to \{0,1\}$.

On peut maintenant décrire la machine qui va calculer la norme. Sur une entrée f et une précision p, la machine demande le module de continuité $\mu_f(p)$ puis demande la valeur de l'oracle supplémentaire sur $A_{\mu_f(p),2\mu_f(p)}$. Elle évalue ensuite f sur les points obtenus, puis fournit à l'oracle la description de la fonction linéaire par morceaux correspondante avant de retourner la valeur obtenue.

Le corollaire 3.1 nous indique alors que le résultat obtenu est à $c2^{-p}$ de la valeur de la norme sur f pour une certaine constante c. Il suffit alors de prendre c en compte pour obtenir la précision souhaitée (en pratique il faut augmenter la précision de $\lceil \log_2(c) \rceil$). Il est alors facile de vérifier que la machine que l'on vient de construire a une complexité d'appel polynomiale. □

3.1.4 Complexité non-déterministe

Comme nous l'avons vu, la norme uniforme et la norme L^1 ont le même ensemble de points importants, ce qui se reflète sous l'angle du calcul par une complexité déterministe exponentielle dans les deux cas. Cependant, comme il sera prouvé dans le théorème 3.7 et la proposition 3.8, la norme uniforme est calculable en temps non-déterministe polynomial (tel que défini dans la définition 2.16), contrairement à la norme L^1.

De même que nous avons fait un lien entre les propriétés de l'ensemble des points importants avec la complexité déterministe des normes, nous allons adapter la notion de dépendance pour capturer les implications analytiques de la complexité non-déterministe.

Comme dit précédemment, Ko [Ko82] a montré que pour toute fonction $f \in \mathcal{C}[0,1]$ calculable en temps polynomial, $\|f\|_\infty$ est un réel calculable en temps polynomial non-déterministe. On sait déjà qu'il est en fait possible (notamment par l'approche de Kawamura et Cook [KC10]) de prouver une version uniforme de ce théorème.

Théorème 3.7. *La norme uniforme $\|\cdot\|_\infty : \mathcal{C}[0,1] \to \mathbb{R}$ est calculable en temps non-déterministe polynomial.*

Preuve

On peut construire une machine $\mathcal{M}$ qui sur une représentation de $f \in \mathcal{C}[0,1]$ et une précision n calcule $m(n+1)$, où m est un module de continuité pour f, puis choisit de façon non-déterministe un point dans $\{\frac{k}{2^{m(n+1)}} \mid 0 \leq k \leq 2^{m(n+1)}\}$ (un tel choix se fait en temps polynomial en $m(n+1)$ et donc en la taille de la représentation de f et n). Enfin, la machine évalue

f à précision $2^{-(n+1)}$ en ce rationnel et retourne sa valeur absolue. La valeur retournée est toujours inférieure à $\|f\|_\infty + 2^{-(n+1)}$ et de plus si $\max_{\alpha\in[0,1]} |f(\alpha)|$ est atteint en β, alors pour $k = \lfloor \beta 2^{m(n+1)} \rfloor$, on a $|\beta - \frac{k}{2^{m(n+1)}}|$ et donc $|f(\beta) - f(\frac{k}{2^{m(n+1)}})| \leq 2^{-(n+1)}$. La valeur retournée par la machine approche $f(\frac{k}{2^{m(n+1)}})$ à $2^{-(n+1)}$ près, et sa valeur absolue approche donc $|f(\beta)| = \|f\|_\infty$ à 2^{-n} près. □

La situation est cependant différente pour la norme L^1. Elle n'est pas calculable en temps polynomial non-déterministe, et on a même le résultat plus fort suivant.

Proposition 3.8. *Aucune norme calculable en temps polynomial non-déterministe n'est plus faible que la norme L^1.*

Preuve

Soit F une norme calculée par une machine $\mathcal{M}$ non-déterministe en temps polynomial. Soit $\mathbf{1}$ la fonction constante de valeur 1. On suppose que $F(\mathbf{1})$ est plus grand que 1 quitte à multiplier F par une constante si nécessaire. Il existe donc au moins un chemin d'exécution de $\mathcal{M}$ sur l'entrée n et une représentation de $\mathbf{1}$ avec module $m = p \mapsto p + n$ qui fournit un rationnel supérieur à $1 - 2^{-n}$. Soit Q_n l'ensemble des rationnels fournis à l'oracle durant cette exécution. La taille de Q_n est donc polynomiale en n. On définit alors g_n par $g_n(x) = \max(1 - 2^n d(x, Q_n), 0)$ pour tout $x \in [0, 1]$. Cette fonction coïncide avec $\mathbf{1}$ sur Q_n et a le même module de continuité m. Le même chemin non-déterministe d'exécution de $\mathcal{M}$ sur cette entrée fournit donc le même résultat, donc $F(g_n) \geq F(\mathbf{1}) - 2^{-n} \geq 1 - 2^{-n+1}$. Hors $\|g_n\|_1 \leq 2^{-n}|Q_n|$ tend vers 0 car $|Q_n|$ est polynomial en n donc g_n converge pour la norme L^1 mais pas pour F. □

Points essentiels et ensembles suffisants

Il est important de remarquer que, contrairement à la complexité déterministe, la complexité non-déterministe est intrinsèquement asymétrique. En effet, par sa définition (définition 2.16) ou par la caractérisation de Ko (remarque 2.9) on peut noter l'implication de la notion de maximum, une machine non-déterministe calculant toujours son résultat par le bas. En particulier, la valeur de la fonction est grande si la valeur retournée sur *une* branche de calcul est grande, alors qu'elle est petite si les valeurs retournées par *toutes* les branches sont petites.

De la même façon, la définition de la dépendance était symétrique : on regardait les variations positives ou négatives de la norme en faisant varier sa fonction d'entrée. Nous allons donc adapter cette notion pour n'étudier que les perturbations de l'entrée qui font significativement décroître la norme.

Alors que la notion de point important visait à définir les points auxquels une machine déterministe devait nécessairement évaluer son entrée, nous définissons la notion purement analytique de points essentiels qui aspire à caractériser les points qui seront nécessairement utilisés sur au moins une des branches non-déterministes.

Définition 3.8 (Points essentiels)**.** *Étant donné une norme F, une fonction $f \in \mathcal{C}[0,1]$ et un point $\alpha \in [0,1]$ on pose*

$$\begin{aligned}\eta_{F,f,\alpha}(n) &= \sup\{l \mid \exists g \in Lip_1, |g| \leq h_{\alpha,\frac{1}{l}} \quad et \quad F(f+g) < F(f) - 2^{-n}\} \\ &= \inf\{l \mid \forall g \in Lip_1, |g| \leq h_{\alpha,\frac{1}{l}} \implies F(f+g) \geq F(f) - 2^{-n}\}.\end{aligned}$$

On dit alors que α est **essentiel** *(relativement à F et f) si il existe $k \in \mathbb{N}$ tel que pour tout $n \in \mathbb{N}$, $\eta_{F,f,\alpha}(n) \geq 2^{\frac{n}{2}-k}$.*

La fonction $\eta_{,,}$ est une notion quantitative, fortement liée à celle de dépendance.

Proposition 3.9.

$$\sup_{f \in \mathcal{C}[0,1]} \eta_{F,f,\alpha}(n) = \mathrm{d}_{F,\alpha}(n).$$

Preuve

Soit $f \in \mathcal{C}[0,1]$ et $l < \eta_{F,f,\alpha}(n)$. Il existe alors $g \in Lip_1$ bornée en valeur absolue par $h_{\alpha,\frac{1}{l}}$ telle que $F(f+g) < F(f) - 2^{-n}$. On a alors $F(g) \geq F(f) - F(f+g) > 2^{-n}$, et donc $\mathrm{d}_{F,\alpha}(n) \geq l$.

Inversement, si $l < \mathrm{d}_{F,\alpha}(n)$ et $g \in Lip_1$ est bornée en valeur absolue par $h_{\alpha,\frac{1}{l}}$ et vérifie $F(g) > 2^{-n}$, on pose $f = -2g$. On a alors $F(f+g) = F(g) = F(f) - F(g) < F(f) - 2^{-n}$, et donc $\eta_{F,f,\alpha}(n) \geq l$. □

Remarque 3.7. *Cela implique que s'il existe une fonction f pour laquelle α est essentiel, alors α est un point important.*

Définition 3.9 (Ensembles suffisants)**.** *Étant donné une norme F, une fonction $f \in \mathcal{C}[0,1]$ et un sous-ensemble A de $[0,1]$, A est n-***suffisant** *relativement à f si pour tout $g \in Lip_1$ nul sur A, $F(f+g) \geq F(f) - 2^{-n}$.*

La notion de points essentiels nous servira principalement à démontrer des propriétés sur les ensembles suffisants. En effet, tout point suffisant doit appartenir à un ensemble n-suffisant pour n assez grand.

Proposition 3.10. *Si A est n-suffisant et $\alpha \notin A$, alors $\eta_{F,f,\alpha}(n) \leq \frac{1}{d(\alpha,A)}$.*

En particulier, pour tout point essentiel α, il existe $k \in \mathbb{N}$ tel qu'il est au plus à distance $2^{-\frac{n}{2}+k}$ de tout ensemble n-suffisant, pour tout $n \in \mathbb{N}$.

Preuve

On pose $l = \frac{1}{d(\alpha,A)}$. Si $g \in Lip_1$ est borné en valeur absolue par $h_{\alpha,\frac{1}{l}}$, alors le support de g est disjoint de A, donc $F(f+g) \geq F(f) - 2^{-n}$, et alors $\eta_{F,f,\alpha}(n) \leq l$. □

Complexité non-déterministe et ensembles suffisants

La propriété suivante fait le lien entre les ensembles suffisants et les calculs non-déterministes.

Proposition 3.11. *Soit F calculée par une machine non déterministe $\mathcal{M}$ et $f \in \mathcal{C}[0,1]$ une fonction de module de continuité m. On considère une représentation de f munie du module $p \mapsto \max(m(p+1), p+1)$ et une branche de l'exécution de $\mathcal{M}$ (sur cette représentation et à précision $n+1$) qui calcule $F(f)$ à précision $2^{-(n+1)}$. Dans ce cas, l'ensemble des appels à l'oracle effectués par $\mathcal{M}$ est n-suffisant.*

Preuve

Soit Q un tel ensemble d'appels, et $g \in Lip_1$ tel que g est nulle sur Q. La fonction $f+g$ a pour module de continuité $p \mapsto \max(m_f(p+1), p+1)$ et est égale à f sur Q. Elle possède donc une représentation qui possède la même branche d'exécution que celle qui a fourni Q (puisque les réponses fournies par l'oracle sont identiques). Donc $F(f+g) \geq F(f) - 2^{-n}$. □

On en déduit le résultat suivant, étant donné qu'une norme non-déterministe polynomiale effectue au plus un nombre polynomial d'appels à l'oracle.

Corollaire 3.3. *Pour tout norme dans $\textsc{nfptime}_{\mathcal{C}[0,1]\to\mathbb{R}}$, et toute représentation de $f \in \mathcal{C}[0,1]$ et $n \in \mathbb{N}$, il existe un ensemble n-suffisant de taille polynomiale (en n et la taille de la représentation de f).*

De plus, en utilisant la proposition 3.10, on obtient un résultat similaire au théorème 3.4 pour le cas déterministe.

Corollaire 3.4. *Pour tout norme dans $\textsc{nfptime}_{\mathcal{C}[0,1]\to\mathbb{R}}$, l'ensemble des points essentiels est de mesure nulle et a même une dimension de Hausdorff nulle.*

Exemple 3.9. *Si $f \in \mathcal{C}[0,1]$ et $\alpha \in [0,1]$, on pose d la distance maximale entre α et tous les points β tels que $|f(\beta)| \geq \|f\|_\infty - 2^{-n}$. On a donc par définition, $\frac{1}{d+2^{-n}} \leq \eta_{\|\cdot\|_\infty,f,\alpha}(n) \leq \frac{1}{d}$.*

En particulier, dans le cas où il existe un unique point maximal α, alors la suite $(\eta_{\|\cdot\|_\infty,f,\alpha}(n))_{n\in\mathbb{N}}$ diverge, alors que $(\eta_{\|\cdot\|_\infty,f,\beta}(n))_{n\in\mathbb{N}}$ est bornée pour tout β différent de α.

S'il y a au moins deux points maximaux, alors $\eta_{\|\cdot\|_\infty,f,\alpha}$ est borné pour tout α.

Un point est donc essentiel pour la norme uniforme si et seulement si il est l'unique point maximal (s'il y en a au moins deux, il suffit d'évaluer la fonction en seulement l'un des deux donc aucun n'est essentiel).

En revanche, si f atteint son maximum (en valeur absolue) en α, alors $\{\alpha\}$ est n-suffisant (relativement à la norme uniforme) pour tout n.

Exemple 3.10. *Si $f \in \mathcal{C}[0,1]$ et $\alpha \in [0,1]$, alors on a $\eta_{L^1,f,\alpha}(n) = 2^{\frac{n}{2}}$. Tout point est essentiel pour la norme L^1, et un ensemble n-suffisant A doit au moins contenir de l'ordre de $2^{\frac{n}{2}}$ points, étant donné que $\mathcal{B}(A, 2^{-\frac{n}{2}})$ doit couvrir l'intervalle $[0,1]$ tout entier, d'après la proposition 3.10.*

Exemple 3.11. *Soit F_q une norme telle que définie dans l'exemple 3.2, et $f \in \mathcal{C}[0,1]$. Tout q_i est essentiel car $\eta_{F,f,q_i}(n) \geq 2^{n-i}$. Si il existe un polynôme P tel que $d(\alpha, q_i) > \frac{1}{P(i)}$ pour tout i, alors $\eta_{F,f,\alpha} \leq P(i)$ pour tout i, et donc α n'est pas essentiel.*

Nous donnons maintenant une caractérisation des normes calculables en temps polynomial non-déterministe relativement à un oracle similaire au théorème 3.6, l'idée étant qu'une telle norme a un nombre relativement faible d'ensembles tels que pour toute entrée au moins l'un d'entre eux est suffisant.

Théorème 3.8. *Une norme F est calculable en temps polynomial non-déterministe relativement à un oracle si et seulement si il existe un polynôme P et une fonction $Q : \{0,1\}^{P(M,n)} \to \mathbb{D}^{P(M,n)}$ tel que pour tout $f \in Lip_1$ vérifiant $\|f\|_\infty \leq 2^M$, il existe $u \in \{0,1\}^{P(M,n)}$ tel que Q_u est n-suffisant pour f et F. Autrement dit, pour tout M, n, il existe un ensemble d'au plus $2^{P(M,n)}$ ensembles de rationnels de taille polynomiale, tel que toute fonction de norme uniforme inférieure à 2^M rend au moins l'un d'entre eux n-suffisant.*

Preuve

On suppose qu'une telle fonction Q existe. On définit alors un oracle A qui permet de calculer Q et la fonction $F_Q : \mathbb{N} \times \{0,1\}^{P(M,n)} \times \mathbb{D}^{P(M,n)} \to \mathbb{D}$ telle que

$$|F_Q(n,u,v) - \inf_{\substack{f \in Lip_1 \\ f=v \text{ sur } Q_u}} F(f)| \leq 2^{-n}.$$

Intuitivement, n indique la précision, u est un certificat à partir duquel l'ensemble Q_u des points d'évaluation est obtenu, et v est la liste des approximations de f à précision 2^{-n} sur Q_u. Sur ces entrées, F_Q fournit une borne inférieure sur $F(f)$ aussi grande que possible, étant donné ces informations.

On construit une machine $\mathcal{M}$ calculant F de façon non-déterministe ainsi : étant donné un oracle représentant $f \in Lip_1$ et une précision n, elle calcule une borne supérieure $2^M \in \mathbb{N}$ pour $\|f\|_\infty$ (en utilisant le lemme 3.1 page 33). Elle choisit ensuite de façon non-déterministe (et en temps polynomial) $u \in \{0,1\}^{P(M,|d|)}$, calcule Q_u grâce à A puis évalue f sur Q_u à précision n. Elle obtient ainsi une liste de valeurs v, qu'elle fournit à A pour obtenir $F_Q(n+1,u,v)$ puis retourne la valeur obtenue.

Si Q_u est $(n+1)$-suffisant pour f alors la valeur obtenue est à distance au plus 2^{-n} de $F(f)$. D'après l'hypothèse initiale, il existe u tel que Q_u est $(n+1)$-suffisant pour f. Pour les autres u', la valeur obtenue est bien plus petite que $F(f) + 2^{-n}$, par définition de F_Q. Cette machine calcule donc bien F en temps polynomial non-déterministe.

Inversement, soit F calculée par une machine $\mathcal{M}$ en temps polynomial non-déterministe relativement à un oracle A. Sur toute représentation φ d'une fonction $f \in \mathcal{C}[0,1]$, toute précision $n \in \mathbb{N}$ et toute suite de choix non-déterministes u (par définition de taille $P(|\varphi|_1, n)$ pour un certain polynôme P), $\mathcal{M}$ effectue un ensemble Q d'appels sur φ de taille au plus $P(|\varphi|_1, n)$. Cet ensemble Q dépend alors uniquement de φ, n et u. Il suffit alors de montrer que l'on peut remplacer la dépendance de Q en φ par $|\varphi|_1$.

Tout $d \in Q$ est de taille $P(|\varphi|_1, n)$, donc les valeurs fournies par l'oracle sont au plus de taille $|\varphi|_1(P(|\varphi|_1, n)$. À chaque u on associe u' formé par u suivi d'une liste v de $P(|\varphi|, n)$ mots de taille $|\varphi|_1|(P(|\varphi|_1, n)$. Le mot u' est alors toujours de taille polynomiale en $|\varphi|_1$ et n.

On considère alors la nouvelle machine $\mathcal{M}'$ qui sur l'oracle φ, la précision n et la suite de choix non-déterministes u' décompose u' en u et la liste v de mots restante, simule $\mathcal{M}$ sur n en suivant le chemin non-déterministe u et lorsqu'une question est posée à l'oracle répond avec le mot suivant dans la liste v. Soient $q_0, \dots, q_k$ les appels à l'oracle effectués. Lorsque $\mathcal{M}$ s'arrête, on interroge φ sur $q_0, \dots, q_k$ et on vérifie que ses réponses coïncident dans l'ordre avec v. Si c'est le cas, $\mathcal{M}'$ répond la même chose que $\mathcal{M}$, sinon elle répond 0.

La machine $\mathcal{M}'$ calcule bien F en temps polynomial non-déterministe, et les appels à l'oracle qu'elle effectue dépendent maintenant uniquement de $|\varphi|_1$, n et v (mais plus de φ directement).

Pour conclure, d'après la proposition 2.5 (page 29), toute fonction $f \in Lip_1$ de norme uniforme bornée par 2^M a une représentation φ de taille bornée par un polynôme en M. Pour la précision $n+1$ est les choix non-déterministes u' de taille polynomiale en n et M, on pose $Q_{u'}$ l'ensemble des appels à l'oracle effectués par $\mathcal{M}'$ (qui est aussi de taille polynomiale en n et M). Alors d'après la proposition 3.11, pour tout $f \in Lip_1$ de norme uniforme bornée par 2^M, il existe u' tel que $Q_{u'}$ est n-suffisant. □

Remarque 3.8. *On peut retrouver le fait que la norme L^1 n'est pas calculable en temps polynomial non-déterministe en observant (cf. exemple 3.10) que cette norme n'a que des ensembles n-suffisant de taille exponentielle.*

En résumé, dans cette section nous avons défini la notion de la dépendance d'une norme en un point et en avons dérivé l'ensemble des points importants relatifs à une norme. Nous avons alors fait le lien entre les propriétés analytiques de cet ensemble avec la complexité déterministe de la norme, en partant du constat que la valeur de la norme dépend des valeurs de son entrée sur l'ensemble des points importants et qu'une norme de faible complexité ne peut évaluer son entrée qu'en un nombre restreint de points.

Nous avons de plus adapté ces notions à un point de vue non-déterministe, et établi un lien similaire entre une propriété analytique des normes et leurs complexité d'appel non-déterministe. Ces deux résultats permettent en particulier de mieux comprendre les implications de la complexité sur la « forme » des fonctions calculées.

3.2 Unique accès à l'oracle

Dans cette section nous étudions un des cas les plus simples, où une machine calculant une fonction quelconque (pas nécessairement une norme) de $\mathcal{C}[0,1] \to \mathbb{R}$ a une complexité d'appel (telle que décrite dans la définition 1.10) bornée par la fonction constante **1**. Plus précisément, une entrée $f \in \mathcal{C}[0,1]$ est représentée (via $\delta_\square$) par une fonction d'approximation et un module de continuité. On supposera dans la suite qu'une fonction est représentée par deux oracles distincts (pour la fonction d'approximation et le module de continuité) et restreindra la machine à n'effectuer qu'au plus un appel à la fonction d'approximation de son entrée pour toute précision n (sans restriction au nombre d'appels au module de continuité).

On peut tout d'abord se pencher sur le problème similaire sur les entiers, plus simple que sur les nombres réels, à savoir : comment caractériser l'ensemble des fonctions $F : (\mathbb{N} \to \mathbb{N}) \to (\mathbb{N} \to \mathbb{N})$ calculables par une machine à oracle faisant au plus un appel sur chaque entrée n. La réponse est simple et intuitive : cela équivaut à pouvoir écrire F comme $F(f)(n) = \varphi(f(\psi(n))$ pour tout $n \in \mathbb{N}$, où $\varphi, \psi : \mathbb{N} \to \mathbb{N}$ sont des fonctions calculables (en particulier, $\psi(n)$ est la question posée à l'oracle par la machine sur l'entrée n). Si de plus F doit être calculable en temps polynomial, alors la caractérisation est similaire, où φ et ψ sont de plus calculables en temps polynomial. Le problème sur les nombres réels exprimé précédemment a une réponse similaire, mais sa preuve en est beaucoup moins immédiate. En particulier, la proposition 3.12 montrera pourquoi ce résultat est plus difficile qu'il n'y paraît.

Théorème 3.9. *Une fonction $F : \mathcal{C}[0,1] \to \mathbb{R}$ est calculable (resp. calculable en temps polynomial) par une machine qui effectue au plus une requête à la fonction d'approximation si et seulement si il existe un nombre réel calculable (resp. calculable en temps polynomial) $\alpha \in [0,1]$ et une fonction calculable (resp. calculable en temps polynomial) $\phi : \mathbb{R} \to \mathbb{R}$ uniformément continue et qui possède un module de continuité calculable (resp. borné par un polynôme) tel que pour tout $f \in \mathcal{C}[0,1]$, $F(f) = \phi(f(\alpha))$.*

Preuve

On peut tout d'abord remarquer que si de tels ϕ et α existent, alors ϕ est unique, et est définie par $\phi(v) = F(\mathbf{v})$, où $\mathbf{v}$ désigne la fonction constante de valeur v. De plus, si F est non constante, α est aussi déterminée de façon unique. En effet, s'il existe f_1, f_2 tels que $F(f_1) \neq F(f_2)$ et $\alpha_1 \neq \alpha_2$ vérifiant le théorème, alors il existe une fonction f_3 (par exemple affine) passant par les points $(\alpha_1, f(\alpha_1))$ et $(\alpha_2, f(\alpha_2))$, et on a alors $F(f_3) = \phi(f_2(\alpha_2)) = F(f_2) \neq F(f_1) = \phi(f_1(\alpha_1)) = F(f_3)$ et donc une contradiction.

On prouve maintenant l'implication directe (la seconde étant immédiate). On définit $M_m[0,1]$ l'ensemble des fonction de $\mathcal{C}[0,1]$ telles que m en est un module de continuité, avec $m : \mathbb{N} \to \mathbb{N}$. Soit $\mathcal{M}$ une machine à oracle calculant F effectuant toujours au plus un appel à l'oracle d'approximation. En particulier, $\mathcal{M}$ est calculable en temps polynomial P d'ordre 2.

Si F est constante le résultat est immédiat et n'importe quel α convient, on supposera donc F non constante dans la suite. En particulier, il existe n_0 et $f_0, g_0 \in \mathcal{C}[0,1]$ tels que $|F(f_0) - F(g_0)| > 2^{-n_0+1}$. Soit m un module de continuité croissant commun à f_0 et g_0 et on pose $M = p \mapsto m(p+1)$, qui est aussi un module de continuité pour f_0 et g_0.

La machine $\mathcal{M}$ sur le module M et à précision $n \geq n_0$ doit donc nécessairement interroger son oracle d'approximation (autrement elle ne distinguerait pas f_0 et g_0, ce qui contredirait l'hypothèse $|F(f_0) - F(g_0)| > 2^{-n_0+1}$). On appelle alors cette requête $(q_n, p_n) \in \mathbb{Q} \times \mathbb{N}$, pour tout $n \geq n_0$ (notons que ces valeurs ne dépendent que de M).

Lemme 3.11. *La fonction ϕ est uniformément continue, et $n \mapsto p_{n+1}$ en est un module de continuité.*

Preuve

Soient $x, y \in \mathbb{R}$ tels que $|x - y| \leq 2^{-p_{n+1}}$. Il existe alors un nombre rationnel r tel que $|x - r| < 2^{-p_{n+1}}$ et $|y - r| < 2^{-p_{n+1}}$. Il existe donc des représentations des fonctions constantes $\mathbf{x}$ et $\mathbf{y}$ de valeur respective x et y qui répondent toutes les deux r à la requête (q_{n+1}, p_{n+1}). La machine

$\mathcal{M}$ fournira donc la même sortie et donc $|F(\mathbf{x}) - F(\mathbf{y})| \leq 2^{-n}$, et donc $|\phi(x) - \phi(y)| \leq 2^{-n}$. □

On notera de plus que ce module de continuité est calculable et que si $\mathcal{M}$ est calculable en temps polynomial, il est de plus borné par un polynôme. Il ne reste plus qu'à déterminer α.

Lemme 3.12. *Soit β un point d'accumulation de la suite $(q_n)_{n\in\mathbb{N}}$. On a alors :*

1. *pour tous $f, g \in M_M[0,1]$, si $f = g$ sur un voisinage de β alors $F(f) = F(g)$.*
2. *pour tous $f, g \in M_m[0,1]$, si $f(\beta) = g(\beta)$ alors $F(f) = F(g)$.*

Preuve

1. Soient $f = g$ de module M égales sur un voisinage U de β. On peut donc construire des représentations ψ_f et ψ_g de f et g telles que $\psi_f(q,p) = \psi_g(q,p)$ pour tout $p \in \mathbb{N}$ et q rationnel dans U.
   Comme β est un point d'accumulation de $(q_n)_{n\in\mathbb{N}}$, il existe un sous-ensemble infini E de $\mathbb{N}$ tel que pour tout $k \in E$, $q_k \in U$. Sur ces $k \in E$, $\mathcal{M}$ ne peut donc pas distinguer les représentations ψ_f et ψ_g et fournit donc un résultat commun. On a donc $|F(f) - F(g)| \leq 2^{-k+1}$, et comme E est infini, on a bien $F(f) = F(g)$.
2. On suppose maintenant que $f(\beta) = g(\beta)$ avec $f, g \in M_m[0,1]$. On peut construire une fonction g_n de module M qui coïncide avec f sur $\mathcal{B}(\beta, 2^{-n})$, est proche de g ($\|g - g_n\|_\infty \leq 2^{-n+1}$) :
   On pose $\beta_n = \max(\beta - 2^{-n}, 0)$ et $\gamma_n = \min(\beta + 2^{-n}, 1)$, puis $\delta_1 = f(\beta_n) - g(\beta_n)$ et $\delta_2 = f(\gamma_n) - g(\gamma_n)$. On a alors $|\delta_1| \leq |f(\beta_n) - f(\beta)| + |f(\beta) - g(\beta)| + |g(\beta) - g(\beta_n)| \leq |\beta_n - \beta| + 0 + |\beta_n - \beta| \leq 2^{-n+1}$. De la même façon, $|\delta_2| \leq 2^{-n+1}$. On définit alors $g_n \in \mathcal{C}[0,1]$ par : $g_n(x) = f(x)$ si $\beta_n \leq x \leq \gamma_n$; $g_n(x) = g(x) + \delta_1$ si $x \leq \beta_n$; et $g_n(x) = g(x) + \delta_2$ si $x \geq \gamma_n$.
   On peut alors vérifier que g_n vérifie les conditions demandées, et d'après le premier point, $F(f) = F(g_n)$. Comme F est continue et que g_n converge vers g pour la norme uniforme, $F(g_n)$ converge vers $F(g)$ et donc $F(f) = F(g)$.

□

On montre maintenant qu'un tel point d'acculumation est unique.

Lemme 3.13. *Si $(q_n)_{n\in\mathbb{N}}$ a au moins deux points d'accumulation, alors F est constante sur $M_m[0,1]$.*

Preuve

Soient β_1 et β_2 deux tels points d'accumulation. Soient $f, g \in \mathcal{C}[0,1]$ de module m. On montre alors que $F(f) = F(g)$, et donc que F est constant.

— On suppose dans un premier temps que $|g(\beta_2) - f(\beta_1)| \leq |\beta_2 - \beta_1|$. On définit alors h comme la fonction affine définie par $h(\beta_1) = f(\beta_1)$ et $h(\beta_2) = g(\beta_2)$. Par hypothèse, est 1-lipschitzienne, et donc de module m (car $m(p) \geq p$) donc d'après le lemme 3.12 on a $F(f) = F(h) = F(g)$.

— On suppose maintenant $|g(\beta_2) - f(\beta_1)| > |\beta_2 - \beta_1|$. Sans perte de généralité (l'autre cas est symétrique), on peut supposer que $f(\beta_1) > g(\beta_2)$. On peut alors définir des fonctions 1-lipschitziennes $h_1, \ldots, h_{2k+2}$ telles que, en posant $h_0 = g$ et $h_{2k+2} = f$ on a pour tout $i \leq k$, $h_{2i+1}(\beta_2) = h_{2i}(\beta_2)$ et $h_{2i+2}(\beta_1) = h_{2i+1}(\beta_1)$. Toujours d'après le lemme 3.12, on a $F(g) = F(h_1) = \ldots = F(h_{2k+2}) = F(f)$.

□

Comme par hypothèse F n'est pas constante, la suite $(q_n)_{n\in\mathbb{N}}$ possède donc un unique point d'accumulation α. D'après le lemme 3.12 on a donc $F(f) = \phi(f(\alpha))$ pour tout f de module m,puisque f coïncide avec la fonction constante $\boldsymbol{f(\alpha)}$ en α.

Il s'agit maintenant de montrer que α est calculable (en temps polynomial si $\mathcal{M}$ l'est). On montre tout d'abord qu'une fonction bornant la vitesse de convergence de $(q_n)_{n\in\mathbb{N}}$ est aussi un module de continuité pour ϕ.

Lemme 3.14. *La fonction $n \mapsto -\log_2|\alpha - q_{n+1}| - 1$ est un module de continuité pour ϕ.*

Preuve

Soit $n \in \mathbb{N}$ et $x, y \in \mathbb{R}$ tels que $|x-y| \leq 2^{-(-\log_2|\alpha-q_{n+1}|-1)} = 2|\alpha - q_{n+1}|$. On définit alors les fonctions affines f et g vérifiant $f(\alpha) = x$, $g(\alpha) = y$, et $f(q_{n+1}) = g(q_{n+1}) = \frac{x+y}{2}$. Comme $|x-y| \leq 2|\alpha - q_{n+1}|$, f et g sont 1-lipschitziennes et donc de module m. Il existe donc des représentations ψ_f et ψ_g de f et g fournissant la même réponse sur la requête (q_{n+1}, p_{n+1}). On a donc $|F(f) - F(g)| \leq 2^{-n}$ et donc $|\phi(x) - \phi(y)| = |\phi(f(\alpha)) - \phi(g(\alpha))| = |F(f) - F(g)| \leq 2^{-n}$.
□

On peut alors montrer que comme ϕ n'est pas constante (auquel cas F le serait aussi), elle ne peut pas avoir de module de continuité sous-linéaire, et donc $(q_n)_{n\in\mathbb{N}}$ doit converger rapidement vers α.

Lemme 3.15. *Il existe $k \in \mathbb{N}$ tel que pour tout $n \in \mathbb{N}$, $|\alpha - q_n| \leq 2^{k-n}$.*

Preuve

On montre tout d'abord que tout module de continuité m_ϕ de ϕ vérifie $m_\phi(n) \geq n - k$ pour un certain $k \in \mathbb{N}$ et pour tout $n \in \mathbb{N}$.

Soient a et b tels que $a < b$ et $\phi(a) \neq \phi(b)$. Il existe alors $k \in \mathbb{N}$ tel que $|\phi(a) - \phi(b)| \geq 2^{-k}(b-a)$. Soit $n \in \mathbb{N}$ tel que $|\phi(a) - \phi(b)| > 2^{-n}$. On a alors nécessairement $b - a \geq 2^{-m_\phi(n)}$ par définition du module de continuité. On divise alors l'intervalle $[a, b]$ en intervalles de longueur $\leq 2^{-m_\phi(n)}$. Comme ϕ varie au plus de 2^{-n} sur chaque intervalle, on a $|\phi(a) - \phi(b)| \leq p2^{-n} \leq (b-a)2^{m_\phi(n)-n}$ et donc $m_\phi(n) \geq n - k$.

Enfin, puisque $m_\phi(n) = -\log_2|\alpha - q_{n+1}| - 1$ est un module de continuité pour ϕ, on a donc bien $|\alpha - q_{n+1}| \leq 2^{k-n-1}$. □

On a donc montré que pour tout module m il existe α tel que $F(f) = \phi(f(\alpha))$ pour tout $f \in M_m[0,1]$. Il reste alors à montrer que cette construction ne dépend pas de m.

Soit m' une fonction croissante telle que $m'(p) \geq p$ et F n'est pas constante sur $M_{m'}[0,1]$. On va donc construire réel $\alpha' \in [0,1]$ tel que $F(f) = \phi(f(\alpha'))$ pour tout $f \in M_{m'}[0,1]$. On suppose par l'absurde que $\alpha' \neq \alpha$. Comme ϕ n'est pas constante, il existe a, b tels que $\phi(a) \neq \phi(b)$. Comme ϕ est continue, a et b peuvent être choisis arbitrairement proches l'un de l'autre et en particulier tels que $|a-b| \leq |\alpha - \alpha'|$. Soit f la fonction affine définie par $f(\alpha) = a$ et $f(\alpha') = b$. Alors f est 1-lipschitzienne, et donc à la fois de module m et m'. On a donc $\phi(f(\alpha)) = F(f) = \phi(f(\alpha'))$ et donc la contradiction $\phi(a) = \phi(b)$.

On a donc bien un unique α tel que pour tout $f \in \mathcal{C}[0,1]$, $F(f) = \phi(f(\alpha))$.

Enfin, pour monter que α est calculable (éventuellement en temps polynomial), il suffit de constater que comme F est continue et non constante, et que les fonctions lipschitziennes sont

denses dans $\mathcal{C}[0,1]$, il existe des fonctions lipschitziennes distinguées par F. Elles possèdent donc un module commun m borné par un polynôme, et d'après le lemme 3.15 α peut être calculé à partir de m (en temps polynomial si $\mathcal{M}$ l'est), ce qui achève la preuve du théorème □

Nous avons montré que si F est calculable, alors ϕ et α le sont aussi, et si F est calculable en temps polynomial, alors de même pour ϕ et α. On peut alors se demander s'il existe un lien plus uniforme entre la complexité de F et celle de α et ϕ. C'est en effet le cas pour ϕ puisque l'on peut définir ϕ par $\phi(v) = F(x \mapsto v)$. Donc si F est calculable en temps $T : (\mathbb{N} \to \mathbb{N}) \to \mathbb{N} \to \mathbb{N}$, alors ϕ est calculable en temps $x \mapsto T(y \mapsto x)$. Il n'en est cependant pas de même pour α dans le cas général. En effet, pour calculer α dans le cas où F n'est pas constante, il faut tout d'abord trouver fonctions distinguée par F. Ce problème est similaire à celui qui consiste à trouver la valeur, dans le cas où elle est unique, annulant une fonction réelle calculable en temps polynomial pour lequel Ko [Ko91] a montré qu'il pouvait être résolu de façon calculable, mais pas de façon efficace.

Proposition 3.12. *Il existe des suites $(\alpha_n)_{n\in\mathbb{N}}$ et $(\phi_n)_{n\in\mathbb{N}}$ telles que α_k n'est pas calculable en temps polynomial uniformément en k et ϕ_k n'est pas constante, mais $F_k(f) := \phi_k(f(\alpha_k))$ est calculable en temps polynomial uniformément en k.*

Preuve

Soit $A \subseteq \mathbb{N}$ un problème décidable en temps 2^k mais pas en temps polynomial. On pose alors $\alpha_k = 1$ si $k \in A$ et $\alpha_k = 0$ si $k \notin A$, et donc $(\alpha_n)_{n\in\mathbb{N}}$ n'est donc pas calculable en temps polynomial en k. On pose de plus $\phi_k(x) = 2^{-2^k} \cdot x$ et $F_k(f) = \phi_k(f(\alpha_k))$.

Pour chaque k, ϕ_k est bien calculable en temps polynomial et donc de même pour F_k. Nous prouvons maintenant que F_k est même calculable en temps polynomial uniformément en k. Autrement dit, il existe une machine $\mathcal{M}$ qui sur une représentation de $f \in \mathcal{C}[0,1]$, k et $n \in \mathbb{N}$ calcule une approximation à 2^{-n} près de $F_k(f)$ en tempos polynomial. L'idée est que si le module de f est petit, alors f varie peu et il n'est pas nécessaire de calculer α_k (il suffit de calculer $f(0)$ qui est alors proche de $f(\alpha_k)$). Si au contraire son module est grand, alors la taille de sa représentation est nécessairement grande, ce qui laisse suffisamment de temps à $\mathcal{M}$ pour calculer α_k.

Soit $f \in \mathcal{C}[0,1]$ représentée par un module f_m et d'une fonction d'approximation $f_\mathbb{Q}$. Pour tous $(n,k) \in \mathbb{N}^2$, la machine $\mathcal{M}$ calcule $f_m(0)$. Si $n \leq 2^k - f_m(0) - 2$, elle demande ensuite $f_\mathbb{Q}(0,0)$ à l'oracle, puis calcule $2^{-2^k} f_Q(0,0)$. Dans le cas contraire, elle calcule α_k (*i.e.* décide si k appartient à A) puis demande $f_\mathbb{Q}(\alpha_k, n)$ et répond $2^{-2^k} f_Q(\alpha_k, n)$ (qui est une bonne approximation de $F_k(f)$.

Par définition du module de continuité, on a $|f(\alpha_k) - f(0)| \leq |f(0) - f(1)| \leq 2^{f_m(0)}$. Dans le premier cas, on a donc $|2^{-2^k} f_\mathbb{Q}(0,0) - 2^{-2^k} f(\alpha_k)| = 2^{-2^k} |f_\mathbb{Q}(0,0) - f(\alpha_k)| \leq 2^{-2^k}(|f_\mathbb{Q}(0,0) - f(0)| + |f(0) - f(\alpha_k)|) \leq 2^{-2^k}(1 + 2^{f_m(0)}) \leq 2^{-2^k + f_m(0) + 1} \leq 2^{-n-1}$. La machine calcule donc une approximation valide de $F_k(f)$ en temps polynomial.

Dans le second cas $n \geq 2^k - \mu_f(0) - 1$. Le temps de calcul de α_k est alors borné par $2^k \leq n + f_m(0) + 1$, qui est bien polynomial en n et $f_m(0)$. □

Cette proposition montre en particulier que le résultat énoncé par le théorème 3.9, ne pouvait être aussi trivial que prévisible.

3.3 Limites de la complexité d'ordre 2

La théorie du calcul sur les objets d'ordre 0, autrement dit les mots finis ou les entiers, a initialement permis de définir les notions de calculabilité et de complexité sur de nombreux ensembles mais n'était pas suffisante pour permettre l'étude d'ensembles indénombrables comme celui des nombres réels. C'est ainsi que sont apparues diverses approches pour étendre cette théorie entre autres aux nombres réels via la théorie du calcul sur les suites infinies, résumée ici dans la section 1.1.

L'approche de Weihrauch (*Type-2 Theory of Effectivity* [Wei00]) permet en effet de fournir des représentations admissibles pour de très nombreux espaces topologiques (plus précisément pour les quotients d'espaces topologiques à base dénombrable [Sch02]) et donc une notion de complexité ayant du sens d'un point de vue topologique. Cependant, l'espace de représentation Σ^ω, bien que suffisant en termes de calculabilité, ne permet pas toujours d'induire une notion de complexité pertinente sur les espaces qu'il représente. En effet, on attend d'une représentation satisfaisante d'un espace de fonctions qu'elle rende la fonction application calculable en temps polynomial, ce qui est impossible à réaliser avec cette approche.

Proposition 3.13 ([Kaw+12a])**.** *Il n'existe pas de représentation $\delta : \Sigma^\omega \hookrightarrow \mathcal{C}[0,1]$ rendant la fonction application (de type $\mathcal{C}[0,1] \times [0,1] \to \mathbb{R}$) calculable en temps polynomial, même restreinte aux fonctions 1-lipschitziennes de $[0,1]$ dans $[0,1]$.*

Kawamura et Cook [KC10] ont donc adopté l'approche consistant à utiliser plutôt $\Sigma^* \to \Sigma^*$ comme espace de représentation et donc la notion de calcul d'ordre 2 décrite ici dans la section 1.2. Ils ont en effet montré que cette méthode est plus significative du point de vue de la complexité, entre autres sur l'ensemble $\mathcal{C}[0,1]$ (cf. théorème 2.2).

Nous allons maintenant montrer que, de même, l'espace de représentation $\Sigma^* \to \Sigma^*$ n'est toujours pas suffisant pour permettre d'induire une définition raisonnable de complexité sur certains espaces encore plus «grands».

On rappelle qu'un ***espace polonais*** est un espace métrique à base dénombrable et qu'un espace ***σ-compact*** est une union dénombrable d'espaces compacts (comme $\mathbb{R}$ ou $\mathcal{C}[0,1]$ par exemple).

Théorème 3.10 ([FH13])**.** *Soit X un espace polonais non σ-compact muni d'une représentation admissible. Alors l'espace $\mathcal{C}[X,\mathbb{R}]$ des fonctions continues de X dans $\mathbb{R}$ n'a pas de représentation $\delta : (\Sigma^* \to \Sigma^*) \hookrightarrow \mathcal{C}[X,\mathbb{R}]$ rendant le temps de calcul de la fonction application (de type $\mathcal{C}[X,\mathbb{R}] \times X \to \mathbb{R}$) borné par une fonction continue de la taille de ses entrées.*

Preuve

Soit X un tel espace, muni d'une représentation admissible δ_X. Supposons qu'il existe une telle représentation $\delta : (\Sigma^* \to \Sigma^*) \hookrightarrow \mathcal{C}[X,\mathbb{R}]$ rendant l'application calculable en temps borné

par $T : (\mathbb{N} \to \mathbb{N}) \times (\mathbb{N} \to \mathbb{N}) \to (\mathbb{N} \to \mathbb{N})$ continue. Autrement dit, pour tout $f \in \mathcal{C}[X, \mathbb{R}]$ muni d'un nom g_f (relativement à δ) et $x \in X$ muni d'un nom g_x (relativement à δ_X), on peut calculer une représentation de $f(x)$ (relativement à δ_C) de taille $T(|g_f|_1, |g_x|_1)$. Donc d'après l'équation 2.1 (remarque 2.1), la fonction continue $T' : (\mathbb{N} \to \mathbb{N}) \to (\mathbb{N} \to \mathbb{N}) \to \mathbb{R}$ définie par $T'(g_1, g_2) = 2^{T(|g_1|_1, |g_2|_1)} + 1$ vérifie

$$\forall g_f \in \mathrm{dom}(\delta), \forall g_x \in \mathrm{dom}(\delta_X), |\delta(g_f)(\delta_X(g_x))| \leq T'(g_f, g_x). \tag{3.1}$$

D'après le théorème de Hurewicz ([Kec95], théorème 7.10, page 39), comme X est un espace polonais qui n'est pas σ-compact, il existe un homéomorphisme $\Phi : (\Sigma^* \to \Sigma^*) \to A$ entre $\Sigma^* \to \Sigma^*$ et un sous-ensemble fermé A de X. Comme Φ est continue et δ_X est admissible, d'après la proposition 2.1, elle est réalisée par une fonction continue $\Psi : (\mathbb{N} \to \mathbb{N}) \to (\mathbb{N} \to \mathbb{N})$ vérifiant donc :

$$\Phi = \delta_X \circ \Psi. \tag{3.2}$$

On définit alors une fonction continue $F : A \to \mathbb{R}$ par :

$$\forall x \in A, F(x) = T'(\Phi^{-1}(x), \Psi(\Phi^{-1}(x))) + 1.$$

C'est une fonction continue d'un sous-ensemble fermé d'un espace normal X vers $\mathbb{R}$, donc d'après le théorème d'extension de Tietze, il peut être prolongé en une fonction continue $F \in \mathcal{C}[X, \mathbb{R}]$. Comme δ est surjective, il existe un nom f_F de F pour δ. On a alors :

$$F(\Phi(f_F)) = T'(f_F, \Psi(f_F)) + 1.$$

D'après l'équation 3.2, $\Psi(f_F)$ est un nom (relativement à δ_X) pour $\Phi(f_F)$, on a donc :

$$F(\Phi(f_F)) \geq F(\Phi(f_F)) + 1$$

d'après l'équation 3.1, ce qui fournit une contradiction. □

Ce résultat montre en particulier que, bien que l'on ait réussi jusqu'à maintenant à parler de complexité pour les fonctions continues définies sur $\mathbb{R}$ ou sur $\mathcal{C}[0, 1]$, ce n'est plus possible à partir d'un ordre supplémentaire. En effet, comme $\mathcal{C}[0, 1]$ est un exemple d'espace polonais non σ-compact on a le corollaire suivant.

Corollaire 3.5. *Il n'existe pas de représentation de $\mathcal{C}[\mathcal{C}[0, 1], \mathbb{R}]$ rendant la complexité de la fonction application bien définie.*

En effet, $\mathcal{C}[0, 1]$ est un exemple d'espace polonais non σ-compact. L'intuition est que $\Sigma^* \to \Sigma^*$ n'est pas un espace de représentation assez grand pour contenir de tels espaces. Autrement dit, un préfixe donné d'un nom dans $\Sigma^* \to \Sigma^*$ ne peut pas réellement contenir suffisamment d'information sur la fonction qu'il représente. De plus, ce théorème reste en particulier valable pour tout espace de représentation homéomorphe à $\Sigma^* \to \Sigma^*$.

Il apparaît donc nécessaire, si on souhaite étudier la complexité d'objets sur de tels espaces, d'utiliser des espaces de représentation plus grands. Après Σ^* et $\Sigma^* \to \Sigma^*$, il semble naturel d'utiliser des ensembles tels que $(\Sigma^* \to \Sigma^*) \to \Sigma^*$ et plus généralement des fonctions de tout ordre. Mais pour pouvoir déduire de ces représentations

des notions et des résultats de complexité il faut tout d'abord définir cette notion sur les espaces de représentation. Or une telle théorie de la complexité à l'ordre supérieur n'est pas encore clairement définie et reconnue pour le moment, comme nous le verrons dans le chapitre 4. L'analyse récursive fournit alors une motivation parmi d'autres pour résoudre le problème de la complexité des fonctions d'ordre supérieur, auquel nous tenterons d'apporter une réponse dans le chapitre 5.

Complexité d'ordre supérieur

Chapitre 4

Modèles de calculs d'ordre supérieur

Contrairement aux fonctions d'ordre 1, il n'existe pas de thèse de Church-Turing à tout ordre. Il existe de nombreux modèles de calculs définissant des ensembles de fonctions calculables très différents. Le premier problème auquel il faut faire face est de savoir quelle classe de fonctions on considère. En effet, certaines approches considèrent seulement les fonctions totales, d'autres toutes les fonctions partielles ou encore les fonctions héréditairement totales (cf. définition 4.1), ce qui rend difficile la comparaison des sous-ensembles calculables définis par chacun des modèles.

Dans ce chapitre nous décrirons quelques-unes de ces approches sans chercher à les comparer en termes de calculabilité (pour cela, le lecteur pourra se référer à l'étude de Longley [Lon00]) ni à être exhaustif. Nous mettrons plutôt en avant les idées sous-jacentes en essayant de comprendre ce que ces approches peuvent apporter en termes de complexité, sachant que peu d'entre elles ont été définies dans ce but.

Kleene a été l'un des premiers à tenter de définir une classe de fonctions partielles calculables d'ordre supérieur à partir de schémas d'opérations, abordés dans la section 4.1. Kleene et Kreisel ont ensuite définis la même classe de fonctions (héréditairement totales cette fois) à partir de définitions très différentes, décrit dans la section 4.2. La sections 4.3 détaillera ensuite la classe BFF, principale classe de complexité à tout ordre. La section 4.4 présentera brièvement les machines d'ordre supérieur de Seth et les difficultés qu'il y a à définir une notion de complexité d'ordre supérieur via cette approche. Enfin, la section 4.5 abordera la classe de fonctions partielles PCF et notamment sa caractérisation par la sémantique des jeux, qui sera le point de départ du chapitre suivant.

On rappelle la définition des types finis :

$$\tau := \mathbb{N} \mid \tau \times \tau \mid \tau \to \mathbb{N}.$$

On considérera aussi parfois (principalement pour une question de simplicité) les types purs :

$$\tau := \mathbb{N} \mid \tau \to \mathbb{N}.$$

Dans la suite, on notera X_τ l'ensemble des fonction de type τ d'une classe de fonctions X.

Comme nous nous intéressons à la complexité, les fonctions concernées ne sont pas nécessairement des fonctions partielles quelconques. En effet, déjà à l'ordre 1, la définition usuelle de complexité est uniquement valable pour les fonctions totales. Il sera alors peut-être nécessaire de se restreindre à des fonctions totales, ou plutôt héréditairement totales dans le sens suivant.

Définition 4.1 (Fonctions héréditairement totales)**.** *Une classe X de fonctions à tout ordre est une classe de fonctions* **héréditairement totales** *si pour tout type σ, toute fonction de $X_{\sigma\to\tau}$ est définie sur X_σ.*

On pourra par exemple considérer la classe S des fonctions héréditairement totales engendrées par $\mathbb{N}$.

4.1 Schémas de Kleene

Kleene [Kle59b ; Kle63] est l'un des premiers à tenter de définir une notion de fonctions récursives à tout ordre à partir de neuf schémas d'opérations (que nous ne détaillerons pas ici). Tout comme le modèle des machines à oracles à l'ordre 2, cette approche manipule les fonctions de manière purement extensionnelle, autrement dit on peut évaluer une fonction mais il est impossible d'obtenir d'informations sur la façon dont cette évaluation est réalisée.

Définition 4.2 ($S_1 - S_9$)**.** *L'ensemble de fonctions Kleene-récursives relativement à une classe X de fonctions héréditairement totales est obtenu à part des neuf schémas d'opérations suivant (les définitions précises ne sont pas nécessaires ici).*

- $\boldsymbol{S_1}$: *Successeur*
- $\boldsymbol{S_2}$: *Constantes*
- $\boldsymbol{S_3}$: *Projections*
- $\boldsymbol{S_4}$: *Comparaison*
- $\boldsymbol{S_5}$: *Récursion primitive*
- $\boldsymbol{S_6}$: *Permutation des arguments*
- $\boldsymbol{S_7}$: *Application d'ordre 1*
- $\boldsymbol{S_8}$: *Application d'ordre supérieur*
- $\boldsymbol{S_9}$: *Invocation d'indice*

La liste des schémas utilisés pour définir une fonction peut être représentée par un indice entier. La puissance de cette algèbre est obtenue en particulier par l'invocation d'indice, qui, étant donné un indice entier et un argument évalue la fonction correspondant à cet indice sur cet argument. Le schéma S_9 est en particulier le seul qui permette de créer des fonctions partielles.

Cette définitions souffre de plusieurs inconvénients en termes de calculabilité. En particulier, elle n'est pas stable par substitution (c.f. [Kle59b]), et le lecteur pourra se rapporter à [GH77] pour plus de détails. Mais surtout, en ce qui nous concerne, cette approche ne semble pas adaptée à une étude de la complexité. En particulier, il ne semble pas facile d'adapter les schéma S_8 et S_9 pour contraindre la complexité de manière non triviale. En particulier, pour restreindre la complexité des fonctions générées par le schéma S_9 il serait nécessaire de limiter l'ensemble des indices possibles

Pour pouvoir parler de complexité, il semble nécessaire d'utiliser un modèle qui mette en avant la façon dont une fonction interagit avec ses arguments (d'avantage que l'invocation d'indice).

Kleene a plus tard remplacé le schéma S_9 par le schéma S_{11} qui permet de définir un opérateur de point fixe à tout ordre et de se passer des indices de fonctions. La classe de fonctions obtenue souffre de moins de faiblesses que la précédente, et est d'ailleurs très proche de PCF (section 4.5) mais ne fournit cependant pas plus de moyens pour aborder le problème de la complexité.

4.2 Fonctions Kleene-Kreisel calculables

Kleene [Kle59a] et Kreisel [Kre59] ont défini à peu près en même temps la même classe de fonctions récursives d'ordre supérieur. Les deux définitions sont basées sur l'idée que toute portion finie d'une fonction continue est déterminée par une portion finie de son entrée. Contrairement à la première approche de Kleene, celle-ci ne concerne que les fonctions héréditairement totales.

4.2.1 Définition de Kleene

L'idée de Kleene est de représenter toute fonction totale continue d'ordre quelconque par une fonction d'ordre 1. Tout d'abord, on peut constater qu'il est possible de représenter une fonction totale continue $F : (\mathbb{N} \to \mathbb{N}) \to \mathbb{N}$ d'ordre 2 par une fonction $\alpha_F : \mathbb{N} \to \mathbb{N}$ ainsi :

$$\forall \vec{v} \in \mathbb{N}^*, \alpha_F(\langle \vec{v} \rangle) = \begin{cases} c+1 & \text{si } \forall g : \mathbb{N} \to \mathbb{N}, (\forall i, g(i) = v_i) \implies F(g) = c \\ 0 & \text{si un tel } c \text{ n'existe pas.} \end{cases} \tag{4.1}$$

(on rappelle que $\langle \vec{v} \rangle$ désigne l'encodage dans $\mathbb{N}$ d'un k-uplet de la forme $(v_1, \ldots, v_k)$).

En effet, si F est continue (relativement aux topologies de la définition 1.4), alors pour toute fonction $f : \mathbb{N} \to \mathbb{N}$, il existe $k \in \mathbb{N}$ tel que $\alpha_F(\langle f(0), \ldots, f(k) \rangle) = F(f) + 1$. Autrement dit, il suffit de fournir suffisamment d'information sur f à α_F pour qu'elle puisse calculer $F(f)$. Lorsque α_F répond 0, cela signifie donc qu'elle a besoin d'un préfixe plus grand de l'entrée pour fournir une réponse.

Une telle fonction α_F est ce que l'on appellera un *associé* pour F. Plus généralement, on dira que α est un associé pour F si pour tout $f : \mathbb{N} \to \mathbb{N}$,

$$\begin{cases} \forall n \in \mathbb{N}, \alpha(\langle f(0), \dots f(k)\rangle) = 0 \text{ ou } F(f) + 1 \\ \exists k \in \mathbb{N}, \forall n \geq k, \alpha(\langle f(0), \dots f(n)\rangle) > 0. \end{cases} \tag{4.2}$$

Autrement dit, soit l'associé fournit la valeur de F sur f, soit il répond 0 pour signifier qu'il a besoin de plus d'information sur f, cependant il existe toujours un préfixe fini de l'associé de f suffisant (ce qui est l'expression de l'idée de continuité).

Pour généraliser cette notion à tout ordre, on définit une fonction d'application $\cdot|\cdot : (\mathbb{N} \to \mathbb{N})^2 \to \mathbb{N}$ entre associés ainsi :

$f|g = f(\langle g(0), \dots g_k\rangle) - 1$ où k est l'entier minimal tel que $f(\langle g(0), \dots g_k\rangle) > 0$.

Autrement dit, on fournit un préfixe fini de g à f jusqu'à ce qu'il ait suffisamment d'information pour fournir une réponse.

Remarque 4.1. *L'opérateur $\cdot|\cdot$ est une fonction partielle calculable par machine à oracle (i.e. il existe une machine qui calcule $f|g$ étant donné des oracles f et g) et le nombre d'appels est $2k + 1$.*

Définition 4.3 (Fonctions Kleene-Kreisel continues)**.** *On définit la notion d'associé pour les fonctions de type pur d'ordre n ainsi que la classe C_n des* **fonctions Kleene-Kreisel continues** *par induction sur n.*

- *$C_0 = \mathbb{N}$, $C_1 = \mathbb{N} \to \mathbb{N}$ et toute fonction de $\mathbb{N} \to \mathbb{N}$ est un associé pour elle-même.*
- *Pour tout $n \geq 1$, on dit que $f : \mathbb{N} \to \mathbb{N}$ est un associé pour $F : C_n \to \mathbb{N}$ si pour tout associé $g : \mathbb{N} \to \mathbb{N}$ de $G \in C_n$, $f|g = F(G)$. On définit alors C_{n+1} comme l'ensemble des fonctions (totales) de $C_n \to \mathbb{N}$ possédant un tel associé.*

Cette définition, restreinte aux types purs pour une raison de simplicité, peut être étendue aux types finis.

Remarque 4.2. *Contrairement à la section précédente, c'est un point de vue intentionnel qui est utilisé ici. Une même fonction peut être représentée par différents associés, et étant donné un associé f, la fonction $f|\cdot$ peut se comporter de façon différente sur des associés d'une même fonction. Par exemple, il peut attendre un préfixe plus ou moins grand avant de fournir une réponse pour différents associés d'une même fonction, mais devra cependant toujours fournir le même résultat lorsqu'il en fournit un.*

La sous-classe RC des fonctions calculables peut être maintenant définie de façon naturelle :

Définition 4.4 (Fonctions Kleene-Kreisel calculables)**.** *Une fonction de C est calculable si elle possède un associé calculable (en tant que fonction de $\mathbb{N} \to \mathbb{N}$). On note alors RC l'ensemble de ces fonctions.*

Remarque 4.3. *Pour toute fonction F de C on peut définir un associé canonique f qui fournit la valeur de F dès qu'elle a suffisamment d'information sur son entrée, tout comme la fonction α_F de l'équation* (4.1)*.*

On ne peut cependant restreindre la définition des associés à cet associé canonique sans changer la notion de calculabilité sous-jacente. En effet, il existe des fonctions de RC pour lesquelles l'associé canonique n'est pas calculable (pour une preuve de ce résultat on pourra se référer à [Nor80]).

Ce modèle ne permet toutefois pas d'induire simplement une notion de complexité intéressante. En effet, il semblerait naturel de dire qu'une fonction d'ordre quelconque est calculable en temps $t : \mathbb{N} \to \mathbb{N}$ si elle possède un associé calculable en temps t. Comme illustré par l'exemple 4.1, il est toujours possible pour un associé de ne pas répondre dès que possible pour gagner du temps.

Exemple 4.1. *La fonction $F = \lambda f.2^{f(0)}$ n'est pas calculable en temps polynomial (ni intuitivement ni selon les définitions du chapitre 1). En effet, l'associé canonique α_0, défini par*

$$\begin{cases} \alpha_0(\langle\rangle) = 0 \\ \alpha_0(\langle v_0, \dots v_k\rangle) = 2^{v_0} \end{cases}$$

n'est pas calculable en temps polynomial. Or elle possède un associé α calculable en temps polynomial, qui attend que l'entrée soit suffisamment grande pour fournir une réponse :

$$\forall v_0, \dots v_k \in \mathbb{N}, \alpha(\langle v_0, \dots v_k\rangle) = \begin{cases} 0 \text{ si } k < 2^{v_0} \\ 2^{v_0} \text{ si } k \geq 2^{v_0} \end{cases}$$

Le test est effectué en temps polynomial en la taille des entrées et dans le deuxième cas, le calcul de 2^{v_0} est bien polynomial en la taille de $\langle v_0, \dots v_k\rangle$.

Ceci est appuyé par le fait que la complexité de l'opérateur $\cdot|\cdot$ (cf. remarque 4.1) est liée à la taille du préfixe lu et non à la taille ou à la complexité de ses arguments.

Remarque 4.4. *Notons que d'après la remarque 4.3 on ne peut pas empêcher ce comportement en forçant un associé minimal qui répond dès que possible. De plus, même la complexité de l'associé canonique n'est pas significative. En effet, l'information sur l'entrée est obtenue par un préfixe de celle-ci et non de façon interactive, comme dans le modèle de machine à oracle. Déjà à l'ordre 2, un associé qui a besoin d'accéder à la $n^{ième}$ valeur de son entrée (comme par exemple pour la fonction $\lambda f.f(n)$) doit nécessairement obtenir toutes les valeurs précédentes, tout comme dans le modèle des machines sur les mots infinis (section 1.1). Ce phénomène est amplifié à l'ordre 3 et va en croissant avec l'ordre.*

4.2.2 Définition de Kreisel

Il existe de nombreuses caractérisations de la classe C, principalement énumérées par Longley [Lon00]. Hyland [Hyl78] a d'ailleurs montré que toutes les constructions

connues permettant d'obtenir C se réduisent toutes au même sous-ensemble RC de fonctions calculables. C'est donc une sorte de thèse de Church-Turing sur C et appuie l'intérêt de cette classe. On peut d'ailleurs vérifier qu'à l'ordre 2, $\mathsf{RC}_{(\mathbb{N}\to\mathbb{N})\to\mathbb{N}}$ est bien égal à l'ensemble de fonctions calculables par machine à oracle tel que défini dans la section 1.2, où on retrouve la même idée de continuité : une fonction est déterminée par ses valeurs sur des portions finies d'information.

Alors que la définition de Kleene adoptait plutôt une définition basée sur la continuité séquentielle, la définition de Kreisel est plus proche de la définition de base de la continuité, c'est-à-dire à partir de voisinages. C'est ici la notion de voisinage qui nous intéresse et non pas la définition précise (pour le moins complexe) des fonctions continues et calculables de Kreisel, nous ne les détaillerons donc pas ici.

Définition 4.5 (Voisinages de Kreisel)**.** *On définit l'ensemble $\mathcal{V}_\tau$ des voisinages associés au type τ par induction sur τ :*

- *$\mathcal{V}_\mathbb{N} = \{\{n\} \mid n \in \mathbb{N}\}$. En particulier, $\mathcal{V}_\mathbb{N}$ engendre la topologie discrète sur $\mathbb{N}$.*
- *Si $U \in \mathcal{V}_\sigma$ et $V \in \mathcal{V}_\tau$, alors on pose $[U \mapsto V] = \{f : \sigma \to \tau \mid f(U) \subseteq V\}$ et $\mathcal{V}_{\sigma\to\tau}$ est alors l'ensemble des intersections finies d'ensembles de ce type :*

$$\mathcal{V}_{\sigma\to\tau} = \bigcup_{\substack{k\in\mathbb{N}\\ U_i\in\mathcal{V}_\sigma\\ V_i\in\mathcal{V}_\tau}} \Big(\bigcap_{i\le k} [U_i \mapsto V_i]\Big).$$

Kreisel définit alors les fonctions continues à partir de fonctions définies sur ces voisinages. Cette définition est en effet équivalente à celle de Kleene, puisque l'on peut remarquer qu'un préfixe fini d'un associé de Kleene décrit une forme particulière de voisinage de Kreisel.

Exemple 4.2. *Si $f : \mathbb{N} \to \mathbb{N}$ est un associé d'une fonction $F : (\mathbb{N} \to \mathbb{N}) \to \mathbb{N}$, vérifiant $f(\langle v_0, \dots v_k\rangle) = n + 1$, alors F appartient au voisinage $[[\{0\} \mapsto \{v_0\}] \cap \dots \cap [\{k\} \mapsto \{v_k\}] \mapsto \{v\}]$.*

Remarque 4.5. *La présentation de Kreisel, bien que plus complexe, permet cependant une description plus fine que celle de Kleene (comme noté dans la remarque 4.4. Par exemple, la fonction de l'exemple 4.1 a besoin de connaître à quel voisinage de la forme $[\{0\} \mapsto \{v_0\}] \cap \dots \cap [\{2^{v_0}\} \mapsto \{v_{2_0^v}\}]$ appartient son entrée, alors que l'approche de Kreisel n'a besoin que du voisinage $[\{0\} \mapsto \{v_0\}] \cap [\{2^{v_0}\} \mapsto \{v_{2_0^v}\}]$, ce qui semble bien plus significatif en termes de complexité.*

Remarque 4.6. *La description par voisinages de Kreisel d'une fonction semble être liée à une notion proche de celle de complexité d'appel non-déterministe étudiée dans la sous-section 3.1.4. En effet, si une fonction est décrite par ses valeurs sur un ensemble de voisinages, alors on peut décider de façon non-déterministe à quel voisinage appartient un argument, et ainsi en déduire sa valeur. La complexité d'appel non-déterministe de cet opération serait alors liée à une certaine notion de taille pour ce voisinage. Par exemple dans le cas de l'exemple 4.1 et de la remarque 4.5, la taille du voisinage serait liée à $|0|$, $|v_0|$ et $|v_{2^{v_0}}|$ (on peut alors dire que l'ensemble $\{0, v_0, v_{2^{v_0}}\}$ est suffisant).*

Bien que robuste en termes de calculabilité, aucune des caractérisations de la classe RC ne semble fournir une notion naturelle de complexité. L'approche de Kreisel, bien que difficilement exploitable dans l'immédiat, apporte cependant déjà une bonne intuition de la mesure de la taille d'une fonction, qui est l'un des principaux ingrédients manquant à une théorie de la complexité à tout ordre.

4.3 Basic Feasible Functionals

La classe BFF a été introduite pour la première fois par Cook et Urquhart [CU93]. C'est l'une des seules tentatives visant à définir une classe de fonctions de complexité analogue à FPTIME à l'ordre supérieur. C'est une classe de fonctions héréditairement totales vérifiant en particulier la plupart des conditions que l'on attend d'une telle classe (énoncées par exemple par Bellantoni [Bel90] et Seth [Set93]), en particulier elle est stable par composition et correspond aux classes de complexité polynomiale bien établies à l'ordre 1 (*i.e.* FPTIME) et 2 (*i.e.* FPTIME_2, définition 1.12).

4.3.1 Définition

Définition 4.6. *Le langage* PV^ω *peut-être décrit simplement comme un* λ*-calcul simplement typé (sur le type de base* $\mathbb{N}$*), avec un symbole de fonction pour chaque* $f \in \text{FPTIME}$ *et un opérateur de récursion* $\mathcal{R} : \mathbb{N} \to (\mathbb{N} \to \mathbb{N} \to \mathbb{N}) \to (\mathbb{N} \to \mathbb{N}) \to \mathbb{N} \to \mathbb{N}$ *d'ordre 2 défini ainsi :*

$$\mathcal{R}(y, F, B, x) = \begin{cases} y \text{ si } x = 0 \\ F(x, \mathcal{R}(y, F, B, \lfloor \frac{x}{2} \rfloor)) \text{ si } |F(x, \mathcal{R}(y, F, B, \lfloor \frac{x}{2} \rfloor))| \leq |B(x)| \\ B(x) \text{ sinon.} \end{cases}$$

La classe de fonctions BFF *est l'ensemble des fonctions définies comme des termes clos de* PV^ω*. On notera de plus* BFF_n *la restriction de* BFF *aux fonctions de type pur d'ordre* n*.*

Par construction, BFF_1 contient FPTIME. C'est en fait une égalité, ce qui n'est pas surprenant puisque cette algèbre de fonctions est très proche de celle définie par Cobham [Cob65] pour caractériser FPTIME (théorème 1.2). De plus, nous avons déjà défini BFF_2 dans le chapitre 1, et le théorème 1.3 exprime que cette classe coïncide avec la classe FPTIME_2 des fonctions calculables en temps polynomial par machine à oracle.

Cette classe vérifie donc les conditions minimales que l'on peut attendre d'une classe de fonctions de complexité raisonnable d'ordre supérieur, en particulier correspondre aux classes FPTIME et FPTIME_2 comme on vient de le voir, et être clos par λ-abstraction et application. De plus, $\mathcal{R}$ est un opérateur raisonnable, puisqu'il contrôle la profondeur de la récursion (en $|x|$) et la taille des valeurs intermédiaires (grâce à B). Autrement dit, il semble raisonnable de demander à ce que tout candidat potentiel à un analogue de FPTIME à tout ordre contienne BFF.

4.3.2 Limites

La première faiblesse de cette définition est qu'elle ne fournit qu'une classe de fonctions et pas une notion de complexité générale, au sens où on ne peut pas en dériver simplement d'autres classes de complexité. La classe BFF est cependant jusqu'à maintenant ce qui s'approche le plus d'un analogue de FPTIME à tout ordre.

On peut de plus se demander si cette classe contient bien toutes les fonctions que l'on attend d'une classe FPTIME_n pour tout n. Irwin, Kapron et Royer répondent à cette question en fournissant une telle fonction qui n'est pas dans BFF_3.

Exemple 4.3 ([IRK02]). *On pose pour tout $n \in \mathbb{N}$, $f_n(y) = 1 \iff y = 2^n$. Les fonctions Φ et Ψ de type $((\mathbb{N} \to \mathbb{N}) \to \mathbb{N}) \times \mathbb{N} \to \mathbb{N}$ sont alors définies ainsi :*

$$\Phi(F, x) = \begin{cases} 0 & \text{si } F(f_x) = F(\lambda y.0) \\ 1 & \text{sinon.} \end{cases}$$

$$\Psi(F, x) = \begin{cases} 0 & \text{si } F(f_x) = F(\lambda y.0) \\ 2^x & \text{sinon.} \end{cases}$$

Intuitivement, la complexité de la fonction Φ est raisonnable. En effet, la fonction f_x est calculable en temps polynomial, uniformément en x. L'évaluation de F sur f_x et la fonction nulle ainsi que la comparaison des valeurs obtenues peut donc se faire en temps raisonnable. On peut alors en conclure que la fonction Φ appartient à BFF_3.

Au contraire, on peut prouver que Ψ n'est pas dans BFF_3. Au premier abord, cela semble justifié par le fait que dans le second cas, le résultat obtenu (2^x) est exponentiel en la taille $|x|$ de l'entrée x. Cependant, on peut remarquer que f_x et la fonction nulle ne diffèrent que sur l'entrée 2^x. Si F parvient à les différencier, c'est donc qu'il les a évaluées en 2^x. Intuitivement, cette première étape a donc déjà pris un temps supérieur à $|2^x|$, et le temps de calcul de Ψ est au plus le double de celui de Φ. Comme la complexité de Φ est raisonnable, celle de Ψ doit l'être aussi.

Le problème vient ici du fait que la classe BFF ne prend pas en compte la «taille» de ses arguments d'ordre 2 et plus. Autrement dit, une fonction qui évalue ses arguments en de grandes entrées, devrait être considérée comme «grande» et ainsi autoriser plus de temps de calcul à une fonction qui la manipule. De plus, l'opérateur de récursion utilisé dans PV^ωest intrinsèquement d'ordre 2, alors qu'un opérateur de récursion à tout ordre (comme le schéma S_{11} de Kleene ou le point fixe de PCF, section 4.5) fournirait une puissance de calcul supplémentaire. Il n'est cependant pas trivial de restreindre un tel opérateur de manière à maîtriser la complexité des fonctions définies.

4.3.3 Caractérisations

Malgré ses faiblesses, BFF est une classe robuste qui possède de nombreuses caractérisations. Notamment celle de Kapron et Cook [CK89] fournissant deux caractérisa-

tions impératives (contrairement à la définition fonctionnelle initiale).

La première est celle des *Typed While Programs*, qui autorise des boucles simples, des assignations et des évaluations de sous-programmes. Les auteurs définissent alors une notion de temps de calcul sur ce langage, et caractérisent BFF par l'ensemble des programmes dont la complexité est bornée par une fonction de BFF. Ce langage simple ne permet cependant pas de capturer toutes les fonctions Kleene-calculables (par exemple, l'itérateur $\lambda Fx.F^{(x)}(\lambda y.y)$), ce qui exclut ici encore d'utiliser ce langage pour définir une notion de complexité générale. Ajouter un opérateur de récursion au langage permettrait bien de capturer tout RC, mais il ne semble pas alors aisé de conserver une notion de temps de calcul intéressante sur ce nouveau langage.

La seconde caractérisation (les *Bounded Typed Loop Programs*) n'utilise pas de notion de temps de calcul, mais limite la complexité en contrôlant le nombre d'itérations et la taille des assignations dans chaque boucle. Plus que le précédent, ce langage permet de concrétiser la notion de complexité définie par BFF.

Enfin, Irwin, Kapron et Royer ont adapté l'idée de *safe recursion* de Bellantoni et Cook [BC92] et de *tier-recursion* de Leivant et Marion [LM93 ; Lei95] pour définir le langage ITLP_2 (*type-2 inflationary tiered loop programs* [IRK01]) caractérisant BFF_2 avant de l'étendre en ITLP [IRK02] à tout ordre pour caractériser BFF.

Ces caractérisations permettent de mieux cerner la classe BFF, mais ne sont pas suffisamment générales pour donner une compréhension globale de la complexité d'ordre supérieur. Pour cela, il semble naturel d'étudier un modèle de machines, comme celui de Seth décrit dans la section suivante, pour lequel définir une notion pertinente de temps de calcul pourrait permettre de définir entre autres des classes de complexité en temps et en espace de façon naturelle.

4.4 Machines à oracle d'ordre supérieur

Dans le chapitre 1 nous avons vu qu'il est possible d'étendre le modèle des machines de Turing usuelles pour obtenir un modèle de machines à l'ordre 2 muni d'une notion de temps de calcul, de taille (des entrées d'ordre 1) et de complexité. Le principe du modèle des machines à oracle est que l'on peut demander à évaluer une entrée d'ordre 1 sur un argument d'ordre 0 en l'écrivant sur un ruban. Si l'on souhaite étendre ce modèle à des oracles de tout ordre fini, il faut se demander comment présenter une entrée d'ordre n à un oracle d'ordre $n+1$, pour tout n entier. Le moyen le plus simple de représenter une fonction d'ordre quelconque est alors d'utiliser une énumération de telles machines et de réprésenter une fonction en entrées par un de ses indices ; c'est cette idée qui est développée par Seth [Set95 ; Set94].

Ces machines sont de simples machines à oracles, les oracles étant des fonctions d'ordre supérieur. Lors d'un appel à un oracle, celui-ci lit le mot sur le ruban spécial et s'évalue sur la fonction calculée par la machine correspondant à l'indice (on ne peut donc évaluer un oracle que sur une entrée calculable). Le coût d'un tel appel est alors unitaire.

Un modèle à base de machines semble être une bonne idée pour obtenir un contrôle sur la complexité, cependant il n'y a ici aucun lien entre l'indice d'une machine et sa complexité. En ce sens, ce modèle est à rapprocher du schéma S_9 de Kleene (section 4.1) qui de la même façon représente une fonction par un indice et ne peut en aucun cas savoir comment l'oracle s'en sert. Seth parvient cependant à définir deux classes de fonctions, l'une d'elles étant égale à BFF, l'autre étant *a priori* strictement plus grande à partir de l'ordre 3. Ceci est entre autres possible en demandant à ce qu'à tout instant, le temps de calcul de la machine soit borné polynomialement en fonction des entrées d'ordre 0 et des réponses déjà fournies par l'oracle. Cela n'est cependant pas suffisant, comme l'illustre l'exemple 4.4.

Exemple 4.4. *La fonction d'ordre 2 $F = \lambda f x.f^{(|x|)}(x)$ itérant son argument f $|x|$ fois est calculable par une machine effectuant des appels de tailles polynomiales en fonction de $|x|$ et de la taille des réponses déjà obtenues. En effet, la machine fournit 0 à l'oracle et obtient $f(0)$, puis lui fournit $f(0)$, qui est bien aussi de taille polynomiale en $|x|$ et $|f(0)|$ (la taille de la réponse déjà obtenue), et ainsi de suite.*

De plus le nombre de tels appels est aussi polynomial en $|x|$, on pourrait donc considérer une telle fonction comme raisonnable. Cependant, cette fonction n'est pas dans BFF_2. En effet, une manière simple de le voir est de composer F avec la fonction polynomiale $\lambda x.2x$ (car BFF est stable par composition). On obtient alors $F(\lambda x.2x)) = \lambda y.y^{|y|}$ qui n'est pas dans $\text{FPTIME} = \text{BFF}_1$.

Seth définit donc une première classe de fonctions en demandant donc des contraintes additionnelles permettant de contrôler aussi la taille des entrées fournies à l'oracle. Il est de plus nécessaire de hiérarchiser de telles machines en fonction des indices qu'elles utilisent pour éviter les phénomènes de point fixe, par exemple lorsqu'une machine utilise son propre indice lors de son exécution. Il obtient de plus une seconde classe (qui est en fait égale à BFF) en demandant de plus à ce que les fonctions utilisées par une machine donnée soient restreintes à un ensemble fini de fonctions paramétrées par les entrées d'ordre 0.

Même si l'une de ces approches permet de caractériser la classe BFF, il semble difficile de justifier la pertinence des contraintes utilisées, et ce modèle ne permet pas de définir de notions de complexité de manière aussi naturelle que les machines de Turing usuelles. De plus, cela ne permet pas d'obtenir une notion générale de complexité uniquement basée sur le temps de calcul et une notion de taille des entrées. Par exemple, pour définir une classe de fonctions calculables en temps logarithmique ou exponentiel ou encore une notion de calcul non-déterministe il serait nécessaire de trouver et de justifier de nouvelles contraintes *ad hoc*.

4.5 PCF et sémantique des jeux

En 1969 (puis publié dans [Sco93]), Scott introduit les bases d'une classe de fonctions partielles monotones par l'intermédiaire de la théorie des domaines. Il en dé-

duit de plus une sous-classe de fonctions dites *effectives*. De nombreux travaux, notamment [Plo77] ont abouti à la définition formelle d'un langage de programmation nommé PCF (*Programming with Computable Functions*), tentant de correspondre à cette classe de fonctions calculables.

4.5.1 PCF

On peut définir PCF comme un λ-calcul simplement typé muni de constantes pour l'arithmétique sur les nombres entiers ainsi que d'un opérateur de point fixe $Y : (\tau \to \tau) \to \tau$ pour tout type τ :

$$\forall f : \tau \to \tau, f(Y(f)) = Y(f).$$

Le langage PCF ne permet cependant pas de décrire toutes les fonctions *effectives* de Scott. C'est en effet un langage séquentiel, *i.e.* il ne contient pas de fonctions intrinsèquement parallèles. Plusieurs modèles de PCF ont d'ailleurs mis en avant ce caractère séquentiel, notamment les *algorithmes séquentiels* de Berry et Curien [BC82 ; Cur93] Le *ou* parallèle, noté por : $\mathbb{B} \times \mathbb{B} \to \mathbb{B}$ et défini par :

$$\mathrm{por}(b_1, b_2) = \begin{cases} 1 & \text{si } b_1 = 1 \text{ ou } b_2 = 1 \\ 0 & \text{si } b_1 = 0 \text{ et } b_2 = 0 \\ \bot & \text{dans les autres cas} \end{cases}$$

est l'exemple le plus connu d'une telle fonction parallèle. En effet, aucune stratégie d'évaluation séquentielle ne permet de caculer une telle fonctions. Une stratégie qui commence par évaluer b_1 ne terminera pas (*i.e.* vaudra $\bot$) sur $(\bot, 1)$, et de manière symétrique si elle commence par évaluer b_2. Pour calculer por il faut nécessairement évaluer les deux arguments en parallèle et ne pas attendre la valeur de l'un d'eux si l'autre retourne 1.

De telles fonctions sont considérées comme calculables par Scott, et Plotkin [Plo77] et Sazonov [Saz76] ont montré qu'elles correspondent en fait exactement au langage PCF $^{++}$, *i.e.* PCF muni de la fonction por et d'un opérateur de recherche parallèle *exists*. Cela illustre bien le fait que même dans un contexte donné, il existe plusieurs notions de calculabilité et fonction du fait que l'on considère toutes les fonctions effectives de Scott ou seulement celles qui sont séquentielles.

À propos de complexité, comme on l'a déjà évoqué, il n'est pas évident que l'on puisse parler de la complexité de toutes les fonctions de PCF (ou de PCF $^{++}$ par exemple) étant donné qu'elles peuvent être partielles. De plus, la définition initiale de PCF ne semble pas particulièrement adaptée à l'étude de la complexité. Le principale problème vient de l'opérateur de point fixe Y dont il semble difficile de maîtriser la complexité. Le langage PV$^{\omega}$peut d'ailleurs être vu comme une restriction de PCF, dont l'opérateur de point fixe a été remplacé par un opérateur de récursion. Mais comme nous l'avons vu, il ne semble pas aisé de choisir un tel opérateur de récursion permettant à la fois de contrôler la complexité des fonctions définies tout en fournissant

une puissance de calcul suffisante. Dans tous les cas, un tel opérateur ne permettrait de définir qu'une seule classe de fonctions, et pas une notion de complexité générale.

4.5.2 Sémantique des jeux

L'un des problèmes majeurs soulevés par PCF est celui de la pleine adéquation (i.e. celui de trouver un modèle de PCF pour lequel deux programmes sont observationellement équivalents si et seulement si ils ont la même interprétation dans ce modèle). En effet, comme énoncé précédemment, le modèle standard de PCF est une abstraction de PCF $^{++}$ et non de PCF.

Hanno Nickau [Nic94 ; Nic+96], Hyland et Ong [HO00] et Abramsky *et al.* [AJM13] ont répondu indépendamment à cette question avec une approche très semblable en utilisant la sémantique des jeux. Nous ne nous intéresseront pas ici à la question de la pleine adéquation en elle-même, mais au modèle utilisé, qui possède par ailleurs de nombreuses applications [Ghi09].

Dans ce modèle, les fonctions sont représentées par des *stratégies* s'opposant dans un jeu. Deux stratégies s'opposant jouent alternativement pour poser des questions (*i.e.* demander de l'information à leur adversaire) ou donner des réponses (*i.e.* fournir de l'information correspondant à une question préexistante).

Remarque 4.7. *On peut rapprocher ce modèle des associés de Kleene. En effet, les fonctions sont ici aussi représentées par des fonctions d'ordre 1 (les stratégies). De plus, l'application d'une fonction à une autre se fait par l'intermédiaire d'un jeu séquentiel (i.e. où les joueurs jouent alternativement) entre deux stratégies. On peut alors voir l'opérateur $\cdot|\cdot$ comme la réalisation d'un jeu entre deux associés jouant alternativement des questions (lorsqu'ils retournent* 0*) ou des réponses (lorsqu'ils retournent une valeur strictement positive).*

Comme on l'a vu (exemple 4.1 et remarque 4.4), le jeu simulé par les associés est inefficace et n'est pas pertinent pour définir une bonne notion de complexité. Au contraire, l'approche plus générale des jeux est un bon candidat pour ce problème car l'information n'est plus obtenue par un préfixe croissant de l'entrée (i.e. une question ne sert qu'à demander plus d'information sur l'entrée), mais par des requêtes précises définies dynamiquement, tout comme dans le cas des machines à oracles.

On peut alors rapprocher cette façon de décrire les entrées avec les voisinages de Kreisel (définition 4.5), une stratégie étant une procédure séquentielle découvrant progressivement un voisinage de plus en plus fin auquel appartient son entrée, et finit par répondre une valeur lorsque ce voisinage est suffisamment précis.

Selon Nickau, une bonne notion de calculabilité d'ordre supérieur doit vérifier trois propriétés, qui sont garanties par construction dans ce modèle.

1. Principe d'information finie : l'exécution d'un calcul ne peut révéler qu'une portion d'information finie sur l'entrée. C'est la caractérisation de la notion de continuité et elle est garantie dans ce contexte par le fait qu'un jeu doit nécessairement se terminer après un nombre fini de questions et de réponses.

2. Séquentialité : Nickau exclut les fonctions parallèles telles que por, ce qui est garanti ici par l'alternance entre les joueurs. Une stratégie ne peut pas poser deux questions de suite sans attendre une réponse de son opposant.
3. Extensionnalité : une stratégie doit nécessairement calculer une fonction, dans le sens où si elle s'oppose à différentes stratégies représentant la même fonction, alors le résultat du jeu doit être le même. Ceci est ici garanti par certaines règles du jeux (en particulier la notion d'*innoncence*) qui empêchent une stratégie de jouer en fonction de questions intermédiaires.

Comme le fait remarquer Nickau, cette approche (contrairement à la définition initiale de PCF) semble donc être un bon point de départ pour définir une notion de complexité d'ordre supérieur, tout du moins dans le cas des fonctions séquentielles. C'est donc sur la sémantique des jeux que nous allons maintenant nous appuyer pour définir une telle théorie, sans toutefois nous limiter aux jeux caractérisant PCF.

Chapitre 5

Sémantique des jeux et complexité

Comme nous l'avons suggéré dans le chapitre précédent, la sémantique des jeux est une approche intéressante du point de vue de la complexité ou plus généralement de la gestion des ressources, certains travaux (par exemple [Mur05 ; LL08 ; Ghi05]) vont d'ailleurs dans ce sens. Il existe plusieurs approches de la sémantique des jeux (notamment via les catégories [HHM07]). Nous choisirons l'approche de la *sémantique des jeux nominale* décrite par Gabbay et Ghica [GG12], qui a l'avantage d'être plus concrète que d'autres par certains aspects, ce qui est préférable dans le cadre de la complexité. La sections 5.1 fournira les définitions principales de la sémantique des jeux nominale. Nous nous intéresserons tout particulièrement aux jeux séquentiels (*i.e.* qui ne contiennent pas de principe d'évaluation parallèle, comme nous l'avons illustré avec la fonction por). À partir de cela, nous définirons dans la section 5.2 de nouvelles notions, notamment la taille d'une stratégie (comme nous l'avons vu, la taille est un ingrédient important pour la complexité), puis nous en déduirons deux définitions globalement équivalentes de la complexité d'une stratégie. En particulier, la seconde sera dérivée d'un modèle de machines généralisant les machines à oracles. Nous en déduirons alors une classe de stratégies calculables en temps polynomial (à partir des polynômes d'ordre supérieur définis par Irwin *et al.* [IRK02]). Après avoir défini ces notions dans un cadre aussi général que possible nous les appliquerons dans la section 5.3 aux *jeux innocents*, c'est-à-dire ceux caractérisant PCF. Cela nous permettra de définir une classe de fonctions calculables en temps polynomial à tout ordre qui correspond aux classes usuelles aux ordres 1 et 2 et étend BFF à tout ordre, tout en incluant des fonctions intuitivement polynomiales, telles que celle de l'exemple 4.3. Enfin, la section 5.4 conclura en proposant des pistes d'applications et de prolongements.

5.1 Sémantique des jeux nominale

L'idée majeure de la sémantique des jeux est de représenter une fonction et ses arguments par deux adversaires qui s'affrontent dans un jeu. Celui-ci sera concrétisé par une *arène*, *i.e.* une structure décrivant les *coups* (autrement dit les interactions) possibles pour chaque joueur. Certains de ces coups seront des *questions*, pour demander de l'information au joueur adverse, d'autres des réponses pour fournir cet information en retour.

Remarque 5.1. *On peut déjà observer ce mécanisme de dialogue alternant questions et réponses dans les modèles de machines connus. Ainsi, une machine de Turing peut être vue comme un processus qui, sur une question initiale (i.e. lorsqu'on la place dans son état initial) demande et obtient son entrée sur un ruban (cette question et cette réponse étant réalisées instantanément) ; enfin, elle fournit une réponse à la question initiale en écrivant une valeur sur le ruban de sortie et en entrant dans son état final.*

Ce phénomène est d'autant plus visible dans le cas des machines à oracles, que ce soient les machines d'ordre 2 (section 1.2) ou les machines de Seth (section 4.4). Le dialogue est alors similaire au précédent, avec de plus des appels à l'oracle. Ainsi, si la machine à oracle calcule la fonction $F : (\mathbb{N} \to \mathbb{N}) \to \mathbb{N}$ sur un oracle calculant la fonction $f : \mathbb{N} \to \mathbb{N}$, alors le dialogue est de la forme suivante, composé de plusieurs sous-dialogues (délimités par une accolade) représentant chacun un appel à l'oracle.

$$
\begin{array}{l}
\textbf{\textit{Question initiale}} \quad : \textit{Que vaut } F \textit{ ?} \\
\quad \vdots \\
\quad \left\{ \begin{array}{l}
\textbf{\textit{Question (machine)}} \quad : \textit{Que vaut } f \textit{ ?} \\
\qquad \textbf{\textit{Question (oracle)}} \quad : \textit{En quel point ?} \\
\qquad \textbf{\textit{Réponse (machine)}} : \textit{En } w_i. \\
\textbf{\textit{Réponse (oracle)}} \quad : \textit{En } w_i, f \textit{ vaut } f(w_i).
\end{array} \right. \\
\quad \vdots \\
\textbf{\textit{Réponse finale (machine)}} : \textit{en } f, F \textit{ vaut } r.
\end{array}
$$

Dans le modèle de machines à oracles, les trois premiers points sont en fait représentés par un appel à l'oracle, le quatrième étant la réponse de l'oracle.

Les jeux sont régis par des règles, par exemple ici chaque réponse est associée à une question qui la précède, et chaque question (à part la première) est justifiée par une question qui la précède. Dans le cas ci-dessus, la question de l'oracle n'aurait pas de sens sans la question de la machine, autrement dit, un oracle ne peut répondre à un appel qui n'a pas été effectué. Ces règles servent donc à décrire les comportements autorisés lors d'un calcul.

On peut alors classer ce genre de dialogue en fonction du nombre maximal de questions imbriquées (deux dans le cas d'une machine de Turing, trois dans le cas d'une machine à oracle). Ce nombre semble alors lié à l'ordre de la fonction calculée, et même si les machines de

Seth calculent des fonctions d'ordre quelconque, les oracles utilisés sont intrinsèquement des fonctions d'ordre 1 (car leurs entrées sont seulement des indices de machines).

Pour généraliser les machines à oracles à l'ordre 3, il faudrait alors pouvoir autoriser les oracles à se comporter comme des machines d'ordre 2, et donc d'exprimer des dialogues de profondeur 4.

5.1.1 Arènes

Définition 5.1 (Arène)**.** *Une* **arène** $\mathcal{A} = (\mathfrak{Q}, \mathfrak{R}, \lambda, \vdash, \mathfrak{I})$ *est définie par :*

- *des ensembles disjoints* $\mathfrak{Q}$ *(***questions***) et* $\mathfrak{R}$ *(***réponses***) qui forment l'ensemble* $\mathfrak{C} = \mathfrak{Q} \cup \mathfrak{R}$ *des* **coups***;*
- *une fonction de* **polarité** $\lambda : \mathfrak{C} \to \{O, J\}$ *associant à chaque coup c un protagoniste (***opposant** *si* $\lambda(c) = O$ *ou* **joueur** *si* $\lambda(c) = J$*). De plus on note* $O^\perp = J$ *et* $J^\perp = O$ *et si* $A \subseteq \mathfrak{C}$*, on notera* A^J *(respectivement,* A^O*) l'ensemble des coups de A appartenant au joueur (respectivement, à l'opposant);*
- *une relation* $\vdash \subseteq \mathfrak{Q} \times \mathfrak{C}$ *définissant une structure sur l'ensemble des coups. Si* $c \vdash c'$ *on dit que c* **autorise** c'*, sachant qu'un coup ne peut être autorisé que par une question du joueur adverse :*

$$q \vdash c \implies \lambda(q) = \lambda(c)^\perp.$$

  On notera de plus $\vdash^*$ *la clôture transitive de* $\vdash$ *et on dira que c autorise récursivement* c' *si* $c \vdash^* c'$*.*
- *un ensemble de* **questions initiales** $\mathfrak{I} \subseteq \mathfrak{Q}^O$ *de l'opposant qui ne sont autorisées par aucune question. On appellera de plus* $\mathfrak{F}$ *l'ensemble des* **réponses finales***, i.e. des réponses autorisées par des questions initiales :*

$$\mathfrak{F} = \{r \in \mathfrak{R}^J \mid \exists i \in \mathfrak{I}, i \vdash r\}.$$

  Intuitivement, les questions initiales sont celles qui permettent d'initier un dialogue, et les réponses finales de le clore.

Dans la suite, c représentera implicitement un coup, q une question, r une réponse, et i une question initiale.

Remarque 5.2. *Une arène peut être vue comme un graphe orienté (défini par la relation* $\vdash$*) biparti (les coups du joueur et les coups de l'opposant). Donc si un coup est dans la composante connexe d'un coup initial, sa polarité est définie par la parité de sa distance à un coup initial. Il n'est donc pas nécessaire de l'expliciter, mais on pourra néanmoins le faire pour plus de clarté.*

Exemple 5.1. *La figure 5.1 présente un exemple d'arène. La relation* $\vdash$ *est représentée par les flèches et les coups par les sommets. L'exposant d'un coup indique sa polarité (J pour le joueur, O pour l'opposant). Enfin, les sommets cerclés représentent les questions initiales. Dans cet exemple,* r_f *et* r_g *sont les réponses finales, et* q_a *est un coup qui n'est pas récursivement autorisé par une question initiale.*

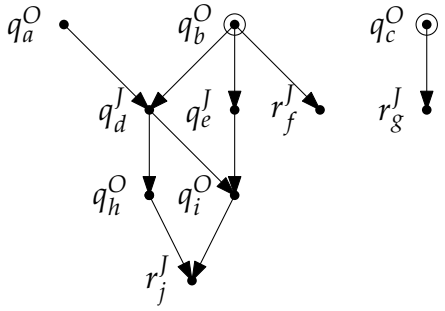

FIGURE 5.1 – Example d'arène.

Les trois exemples suivants décrivent des arènes qui représentent les types purs d'ordre 0 à 2 et fournissent des intuitions sur les stratégies correspondantes qui seront formalisées plus loin.

Exemple 5.2 (Arène $\mathcal{A}_{\mathbb{N}}$)**.** *La figure 5.2 décrit un exemple d'arène permettant de représenter les entiers. Un joueur représentant un entier n jouera la réponse r_n si on lui demande sa valeur (i.e. en posant la question q). On peut considérer que c'est cette représentation qui est utilisée*

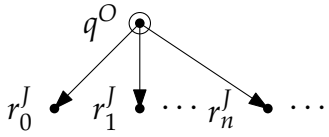

FIGURE 5.2 – Représentation de l'arène $\mathcal{A}_{\mathbb{N}}$.

par une machine de Turing calculant une constante $n \in \mathbb{N}$: lorsqu'on la place dans son état initial, on simule la question q, puis la machine écrit la représentation binaire de n sur son ruban puis entre dans son état final, ce qui simule la réponse r_n.

Exemple 5.3 (Arène $\mathcal{A}'_{\mathbb{N}}$)**.** *Il existe cependant d'autres manières de représenter les entiers. Par exemple, lorsque l'on définit certaines classes de complexité sous-exponentielles comme* LOGTIME *(i.e. dans le cas des machines de Turing à accès direct [2]) on représente un entier par une fonction qui le décrit bit à bit, autrement dit par un oracle qui, si on lui fournit un entier p, répond la valeur du $p^{ième}$ bit de l'entrée. Dans ce cas, on peut représenter l'interface entre la machine et son entrée par l'arène $\mathcal{A}'_{\mathbb{N}}$, décrite par la figure 5.3. Lorsqu'on lui demande la valeur de l'un de ses bits (question q'), la machine calculant l'entier peut demander (question q) à son oracle la valeur de l'indice n (réponse r_n) du bit demandé, puis fournit sa valeur (r_0 ou r_1).*

Exemple 5.4 (Arène $\mathcal{A}_{\mathbb{N}\to\mathbb{N}}$)**.** *L'arène $\mathcal{A}_{\mathbb{N}\to\mathbb{N}}$ (figure 5.4), très similaire à l'arène $\mathcal{A}'_{\mathbb{N}}$ (figure 5.3), formalise l'interface usuelle d'une machine de Turing. L'orsqu'on l'exécute (i.e. on*

2. *i.e. random-access Turing machines*

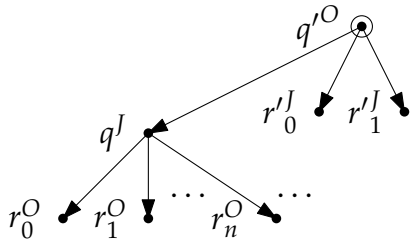

FIGURE 5.3 – Représentation de l'arène $\mathcal{A}'_{\mathbb{N}}$.

lui pose la question q'), elle peut demander à évaluer son entrée (i.e. poser la question q) qui lui est fournie sur un ruban (i.e. on lui fournit la réponse r_n correspondante) avant de terminer en écrivant le résultat sur le ruban de sortie (i.e. répondre une valeur $r'_{n'}$).

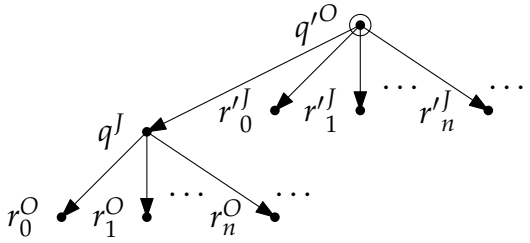

FIGURE 5.4 – Représentation de l'arène $\mathcal{A}_{\mathbb{N}\to\mathbb{N}}$.

Exemple 5.5 (Arène $\mathcal{A}'_{\mathbb{N}\to\mathbb{N}}$)**.** *De même que dans l'exemple 5.3, il existe d'autres représentations possibles du type $\mathbb{N} \to \mathbb{N}$. Par exemple, le dialogue suivant semble être plus approprié pour décrire l'interaction entre une machine de Turing (représentée par le joueur) calculant une fonction f et son entrée représentant l'entier n :*

Opposant (question q_n) *Que vaut f en n ?*

Joueur (réponse $r_{f(n)}$) *En n, f vaut $f(n)$.*

Cette interaction est formalisée par l'arène $\mathcal{A}'_{\mathbb{N}\to\mathbb{N}}$, figure 5.5. Dans ce cas, l'opposant

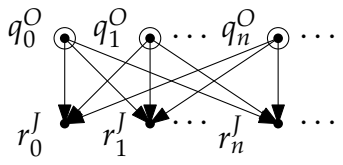

FIGURE 5.5 – Représentation de l'arène $\mathcal{A}'_{\mathbb{N}\to\mathbb{N}}$.

fournit directement la valeur de l'entrée au joueur, de même qu'une machine de Turing possède immédiatement son entrée lorsqu'on la place dans son état initial.

Exemple 5.6 (Arène $\mathcal{A}_{(\mathbb{N}\to\mathbb{N})\to\mathbb{N}}$)**.** *De la même façon, on peut représenter l'interface d'une machine d'ordre 2 par l'arène $\mathcal{A}_{(\mathbb{N}\to\mathbb{N})\to\mathbb{N}}$, figure 5.6. Celle-ci peut demander à évaluer son argument (en posant la question q'), celui-ci se comportant alors comme une fonction d'ordre 1. Ce dernier peut à son tour demander à connaître son argument (en jouant q), la machine lui fournissant une réponse sur le ruban d'appel (autrement dit, joue une réponse r_n) ; l'oracle peut alors répondre une valeur $r'_{n'}$. Après un nombre fini de telles requêtes à l'oracle, la machine peut alors donner son résultat final en répondant l'un des $r''_{n''}$.*

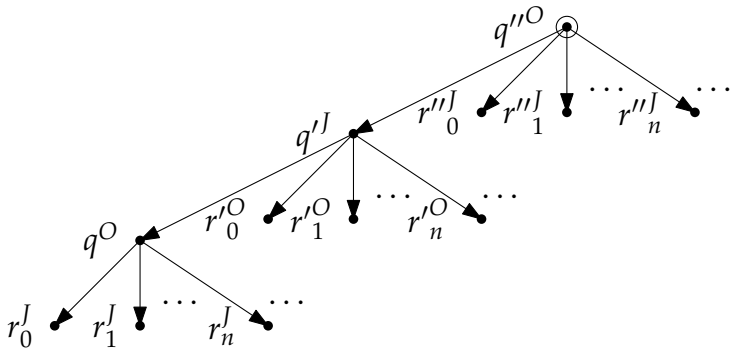

FIGURE 5.6 – Représentation de l'arène $\mathcal{A}_{(\mathbb{N}\to\mathbb{N})\to\mathbb{N}}$.

5.1.2 Parties

Comme illustré dans la remarque 5.1, un coup n'a de sens que s'il est joué après une question qui l'autorise (ce qui est formalisé par la relation $\vdash$). Cependant, si plusieurs questions peuvent autoriser un même coup, il peut être nécessaire de désigner précisément à laquelle il fait référence. Pour cela, plusieurs solutions sont possibles, mais nous suivrons ici l'approche de la sémantique des jeux nominale (décrite par Gabbay et Ghica [GG12]), qui identifie chaque coup joué par un nom. On peut alors définir formellement les *parties*, autrement dit les suites de coups joués par des adversaires, où chaque coup est nommé et nomme le coup auquel il fait référence.

Définition 5.2 (Parties)**.** *Soit $\mathcal{A}$ une arène et $\mathfrak{N}$ un ensemble, appelé ensemble des* **noms***. Si c est un coup et α, β deux noms, alors $c\alpha[\beta]$ désigne le* **coup nommé** *c auquel on a donné le nom β et faisant référence à un autre coup nommé dont le nom est α. Une partie p est alors une liste finie de coups nommés telle que tout nom désigne de façon unique un coup nommé :*

$$p = \dots c_1\alpha_1[\beta_1], \dots c_2\alpha_2[\beta_2] \dots \implies \beta_1 \neq \beta_2. \tag{5.1}$$

On dira alors que c' **justifie** *c dans la partie p si c' apparaît avant c dans p et c fait référence à c' :*

$$p = \dots c'\alpha[\beta] \dots c\beta[\gamma] \dots$$

La condition d'unicité (équation 5.1) implique qu'un tel $c'\alpha[\beta]$ est unique. On dit alors que $c\beta[\gamma]$ est **justifié** *par $c'\alpha[\beta]$.*

Par la suite, $\alpha, \beta, \ldots$ dénoteront implicitement des noms et $p, p' \ldots$ des parties. Pour tout coup (ou coup nommé) c et toute partie p, on pourra noter $c \in p$ si c apparaît dans p. Enfin, on pourra éventuellement omettre les noms dans les parties lorsque les justifications sont implicites ou inutiles.

Définition 5.3. *Dans une partie p, une question $q\alpha[\beta]$ est dite* **fermée** *si elle justifie une réponse qui la succède dans p :*

$$\exists r \in \mathfrak{R}, \gamma \in \mathfrak{N}, p = \ldots q\alpha[\beta] \ldots r\beta[\gamma] \ldots$$

Dans ce cas on dira que $r\beta[\gamma]$ répond à cette question, et dans le cas contraire on dira que cette question est **ouverte**.

Pour définir un jeu il faut définir l'ensemble des coups autorisés à tout instant, autrement dit l'ensemble des parties possibles, ce qui permet de spécifier comment le joueur et son opposant peuvent interagir. Les définitions suivantes fournissent des exemples de restrictions possibles (autrement dit de règles) sur les parties.

Définition 5.4 (Partie justifiée)**.** *Une partie p est* **justifiée** *si tout coup non initial est justifié par un coup qui le précède dans p :*

$$c \notin \mathfrak{I} \wedge p = p' c\beta[\gamma] \ldots \implies \exists c', \alpha, c' \vdash c \wedge c'\alpha[\beta] \in p'.$$

Remarque 5.3. *Dans une partie justifiée, les coups initiaux n'ont pas besoin de justification, on peut donc les noter sous la forme $i[\alpha]$.*

Définition 5.5 (Partie bien ouverte)**.** *Une partie p est* **bien ouverte** *si p commence par un coup initial et n'en contient pas d'autre :*

$$p = p' i\alpha[\beta] \ldots \implies p' = \varepsilon$$

Remarque 5.4. *Une partie justifiée peut être représentée par ses justifications sous forme de graphe orienté acyclique. Si de plus elle est bien ouverte, alors c'est un arbre. De plus, dans une partie bien ouverte, le coup inital n'a pas besoin de justification, on pourra donc l'omettre par la suite.*

Dans une partie alternante, les joueurs jouent l'un après l'autre. C'est en particulier cette propriété qui induit une notion de séquentialité dans le calcul. En effet, un processus parallèle peut être caractérisé par la possibilité de poser plusieurs questions successivement, sans attendre de réponse immédiate de la part de son opposant. Par exemple, le comportement du *ou parallèle* por décrit dans le chapitre précédent ne peut être représenté par une partie alternante.

Définition 5.6 (Partie alternante)**.** *Une partie p est* **alternante** *si deux coups successifs appartiennent à des joueurs différents :*

$$p = \ldots c\alpha[\beta], c'\gamma'[\delta] \ldots \implies \lambda(c) = \lambda(c')^{\perp}.$$

Remarque 5.5. *Pour une partie alternante justifiée, la polarité du dernier coup est donnée par la parité de la longueur de la partie.*

Définition 5.7 (Partie à portée stricte)**.** *Une partie p est* **à portée stricte** *si une fois fermée, une question ne peut plus permettre de justifier un coup.*

$$p = \dots q\alpha[\beta], \dots, r\beta[\gamma], p' \implies \nexists c, \delta, c\beta[\delta] \in p'$$

En particulier, on ne peut répondre deux fois à la même question.

Remarque 5.6. *La condition d'unicité des noms dans une partie n'est pas restrictive pour la suite de ce chapitre. Elle peut être supprimée dans certains cas, ce qui permet de réutiliser des noms lorsque cela ne prête pas à confusion. C'est le cas en particulier si les parties sont à portée stricte, puisqu'une fois qu'une question est fermée, plus aucun coup ne peut y faire référence et on peut alors réutiliser son nom pour nommer un nouveau coup dans la suite de la partie (on peut rapprocher cette idée de celle de gestion dynamique de la mémoire dans un programme). Le lecteur pourra se référer à [GC11] pour une étude sur le problème de la réutilisation des noms.*

Définition 5.8 (Partie bien parenthésée)**.** *Une partie p est* **bien parenthésée** *si les questions et les réponses forment un bon parenthésage :*

$$p = \dots, q_1\alpha[\beta_1], \dots, q_2\beta_1[\beta_2], p', r_1\beta_1[\gamma_1] \dots \implies \exists, r_2, \gamma_2, r_2\beta_2[\gamma_2] \in p'.$$

Cela implique que l'on ne peut répondre à une question avant d'avoir répondu à toutes les questions qu'elle a justifiées.

Remarque 5.7. *Dans le cas où seules des parties justifiées sont autorisées, on peut considérer que l'arène ne contient pas de coups qui ne sont pas héréditairement autorisés par une question initiale, puisqu'ils ne peuvent pas apparaître dans une partie. Dans ce cas, l'arène est un graphe orienté biparti dont les racines sont les questions initiales, et les réponses finales sont des feuilles.*

5.1.3 Stratégies

On peut maintenant définir les *stratégies*, autrement dit les décisions déterministes prises par le joueur en fonction de ce qu'il connaît sur le jeu en cours.

Définition 5.9 (Stratégie)**.** *Étant donné une arène, un ensemble de noms et un ensemble $\mathfrak{P}$ de parties, une stratégie $s : \mathfrak{P} \hookrightarrow \mathfrak{C}$ est une fonction partielle qui, étant donné une partie fournit un coup du joueur qui prolonge la partie en une partie valide :*

$$\forall p \in \mathfrak{P}, p, s(p) \in \mathfrak{P}.$$

Par extension, on dira qu'une stratégie est justifiée (respectivement bien ouverte, bien parenthésée, etc.) si elle prolonge les parties justifiées (respectivement bien ouvertes, bien parenthésées, etc.) en parties justifiées (respectivement bien ouvertes, bien parenthésées, etc.).

Remarque 5.8. *Une stratégie jouant sur des parties justifiées alternantes n'est définie que sur les parties terminant sur un coup de l'opposant (i.e. de longueur impaire).*

Exemple 5.7. *Sur l'arène $\mathcal{A}_{\mathbb{N}}$ définie dans l'exemple 5.2, on dit qu'une stratégie s représente un entier n si $s(q) = r_n$ (les noms sont omis, r_n justifiant implicitement q).*

De même, sur $\mathcal{A}'_{\mathbb{N}}$, s représente n si :

$$\begin{cases} s(q') = q \\ \forall i \in \mathbb{N}, s(q', q, r_i) = r'_{b_i}, \text{ où } b_i \text{ est le } i^{\text{ième}} \text{ bit de l'écriture binaire de } n. \end{cases}$$

Exemple 5.8. *Sur l'arène $\mathcal{A}_{\mathbb{N}\to\mathbb{N}}$ définie dans l'exemple 5.4, on peut définir qu'une stratégie s représente une fonction f si :*

$$s(q') = r'_n, \text{ auquel cas } f = \lambda x.n$$
$$ou \begin{cases} s(q') = q \\ s(q', q, r_n) = r'_{f(n)} \end{cases}$$

Autrement dit, la stratégie répond immédiatement la valeur de la fonction si elle n'a pas besoin de connaître son argument (elle est donc constante), ou bien elle demande la valeur n de son argument et répond $f(n)$. On peut éventuellement autoriser à ce qu'elle demande son argument plusieurs fois, à condition qu'elle finisse par fournir une réponse finale.

De même, dans l'arène $\mathcal{A}'_{\mathbb{N}\to\mathbb{N}}$, on dit que la stratégie s représente la fonction $f : \mathbb{N} \to \mathbb{N}$ si pour tout $n \in \mathbb{N}$, $s(q_n) = r_{f(n)}$.

Exemple 5.9. *Sur l'arène $\mathcal{A}_{(\mathbb{N}\to\mathbb{N})\to\mathbb{N}}$ définie dans l'exemple 5.6, on dit qu'une stratégie s représente une fonction F d'ordre 2 si (tout comme une machine à oracle) elle répond immédiatement (auquel cas F est constante), ou bien elle demande à évaluer son entrée en divers points avant de répondre. Autrement dit, un jeu pourra ressembler à :*

$$q'', q', q, r_{v_0}, r'_{v'_0}, \ldots q', q, r_{v_k}, r'_{v'_k}, r''_v$$

où chaque q' est justifié par la question initiale q'', chaque question q est justifiée par l'occurrence de q' qui la précède immédiatement, et chaque réponse r'_i (respectivement r_i) répond à la question q' (respectivement q) la plus proche qui la précède. On retrouve ici le comportement d'une machine à oracle : la machine (le joueur) interroge son oracle (l'opposant) successivement sur chaque v_i et obtient à chaque fois une valeur v'_i. Après un nombre fini d'appels, elle doit retourner une valeur v'' et s'arrêter. On remarque alors qu'une telle partie est composée (en plus de la question initiale et de la réponse finale) d'une succession de parties dans l'arène $\mathcal{A}_{\mathbb{N}\to\mathbb{N}}$ dont la polarité a été inversée, puisque c'est l'opposant (l'oracle) qui prend alors le rôle du joueur.

Les exemples précédents seront davantage formalisés dans les exemples 5.12, 5.13 et 5.14.

5.1.4 Sous-arènes

Nous avons illustré dans l'exemple 5.9 la dualité entre le joueur et l'opposant, celui-ci se comportant comme un joueur dans une arène plus petite. Nous allons formaliser cette idée par la notion de sous-arènes d'une arène, une sous-arène définissant l'interface entre une stratégie sur l'arène complète avec l'un de ses arguments.

Définition 5.10 (Sous-arènes)**.** *Étant donné une arène $\mathcal{A}$ et un coup initial i, la sous-arène $\mathcal{A}^i$ est obtenue à partir de $\mathcal{A}$ en supprimant i et tous les coups qui ne sont pas héréditairement autorisés par i (i.e. en gardant ceux, sauf i, qui sont accessibles depuis i). Les polarités sont inversées et les questions autorisées par i dans $\mathcal{A}$ sont maintenant initiales dans $\mathcal{A}^i$.*

$$\mathcal{A}^i = (\{q \in \mathfrak{Q} \setminus \{i\} \mid i \vdash^* q\}, \{r \in \mathfrak{R} \mid i \vdash^* a\}, \lambda^{\perp}, \vdash, \{q \in \mathfrak{Q} \mid i \vdash q\})$$

Remarque 5.9. *Il se peut qu'il y ait des coups inaccessibles dans la sous-arène, par exemple les réponses exclusivement autorisées par i. Si on ne considère que des parties justifiées sur cette arène on peut alors les supprimer d'après la remarque 5.7.*

Exemple 5.10. *La figure 5.7 présente la sous-arène associée à la question initiale q_b de l'arène décrite par la figure 5.1 (on a supprimé la réponse r_f car elle était inaccessible).*

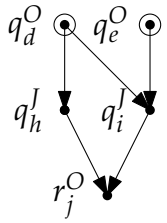

FIGURE 5.7 – Sous-arène associée à q_b de l'arène de la figure 5.1

Exemple 5.11. *L'arène $\mathcal{A}_{\mathbb{N}}$ (exemple 5.2) est la sous-arène $\mathcal{A}^{q'}_{\mathbb{N}\to\mathbb{N}}$ de l'arène $\mathcal{A}_{\mathbb{N}\to\mathbb{N}}$ (exemple 5.4), qui est elle-même la sous-arène $\mathcal{A}^{q''}_{(\mathbb{N}\to\mathbb{N})\to\mathbb{N}}$ de l'arène $\mathcal{A}_{(\mathbb{N}\to\mathbb{N})\to\mathbb{N}}$ (exemple 5.6).*

5.1.5 Jeux séquentiels

Nous allons maintenant nous limiter à ce que nous allons appeler les *jeux séquentiels effectifs*. En particulier les parties seront alternantes, il ne sera donc pas possible pour une stratégie de poser deux questions successivement sans attendre de réponse de son opposant, ce qui exclut de pouvoir représenter des fonctions parallèles comme por. En effet, nous allons définir un processus d'application entre deux stratégies en les faisant jouer tour à tour, interdisant de fait les interactions asynchrones.

Un ***jeu*** est défini par une arène, un ensemble de noms, de parties et de stratégies. Par la suite, $\mathcal{J}$ désignera implicitement un jeu.

Nous allons donc maintenant nous restreindre à une certaine classe de jeux pour lesquels les stratégies représentent le comportement de programmes séquentiels.

Définition 5.11 (Jeu séquentiel effectif)**.** *Un* **jeu séquentiel effectif** *est un jeu $\mathcal{J}$ dont l'arène $\mathcal{A}$ est acyclique et a un nombre fini de questions initiales, et dont les parties sont alternantes, justifiées et bien ouvertes.*

Il est de plus muni :

1. *de jeux séquentiels effectifs $\mathcal{J}^i$ pour chaque sous-arène $\mathcal{A}^i$ de $\mathcal{A}$, appelés* **sous-jeux** *(cette définition est récursive, le cas de base étant lorsqu'il n'y a pas de sous-arène, donc de sous-jeux) ;*
2. *d'une notation* enc *(cf. définition 2.1) des noms et des coups ;*
3. *d'une fonction* vue *fournissant une partie dans un sous-jeu terminant par un coup de l'opposant étant donné une partie dans $\mathcal{J}$ terminant par un coup du joueur (dans $\mathcal{J}$) et d'une fonction* $\text{vue}^\perp$ *fournissant un coup nommé de l'opposant dans $\mathcal{J}$ étant donné une telle partie dans $\mathcal{J}$ et un coup nommé du joueur dans un sous-jeu :*

$$\forall p \in \mathfrak{P}, \text{vue}(p) \in \mathfrak{P}^i$$

et

$$\forall p \in \mathfrak{P}, \forall c \in \mathfrak{C}^i, p, \text{vue}^\perp(p, c) \in \mathfrak{P}.$$

Un tel jeu est effectif en ce sens : α est le nom d'un coup en position n dans une partie $p \in \mathfrak{P}$, alors la taille de son encodage est plus petite que n. De plus, les fonctions vue *et* $\text{vue}^\perp$*, en tant que fonctions sur les mots (via* enc*) sont calculables en temps polynomial.*

Dans un jeu séquentiel effectif, une stratégie va pouvoir s'opposer à une stratégie dans un sous-jeu selon l'interaction qui sera définie dans la définition 5.13. Les fonctions vue et $\text{vue}^\perp$ vont permettre de passer d'une partie pour la première stratégie à la partie correspondante pour la stratégie dans le sous-jeu, et inversement (voir la définition 5.13 pour plus de précisions). L'hypothèse sur leur complexité n'est pas réellement restrictive (intuitivement, de telles fonctions ont une complexité linéaire), ni nécessaire, mais son utilité sera soulignée par la remarque 5.18. Enfin, l'encodage permet un point de vue effectif sur de tels jeux, et permettra en particulier de définir les stratégies calculables (cf. définition 5.12). Selon la remarque 5.11, il existe un encodage canonique vérifiant les hypothèses requises, on l'utilisera donc implicitement dans la suite s'il n'est pas précisé.

Remarque 5.10. *L'arène d'un jeu séquentiel effectif est donc un graphe acyclique orienté de profondeur finie. Le nombre de nœuds internes (i.e. de questions) est fini, mais le nombre de feuilles (incluant les réponses) est potentiellement infini, mais dénombrable (car il est muni d'une notation). En particulier, une telle arène a un nombre fini de questions initiales, et donc un nombre fini de sous-arènes.*

Cette définition est donc bien fondée, le cas de base étant celui du jeu vide $\mathcal{J}_\emptyset$ muni de l'arène vide, qui n'a pas de sous-jeu et dont la seule stratégie est la stratégie $s_\perp$ définie nulle part.

Remarque 5.11. *L'hypothèse sur l'encodage est peu restrictive, puisqu'il est toujours possible de nommer les coups par l'écriture binaire de l'indice de leur position dans la partie. Le choix d'un tel encodage est surtout important si l'on souhaite étudier des classes de complexité sous-linéaires (auquel cas il faudrait aussi appliquer la remarque 5.6), ce que nous n'étudierons pas ici. Elle implique aussi qu'il n'existe qu'un nom pouvant dénoter un coup initial (appelons-le α_0) dont l'encodage est le mot vide.*

Exemple 5.12. *Le jeu $\mathcal{J}_{\mathbb{N}}$ est défini sur l'arène $\mathcal{A}_{\mathbb{N}}$ (cf. figure 5.2). Son unique sous-jeu est le jeu vide $\mathcal{J}_{\emptyset}$. Les fonctions* vue *et* $\text{vue}^{\perp}$ *sont donc définies nulle part, et les seules parties valides sont les préfixes des parties de la forme $q[\alpha_0], r_n\alpha_0[\alpha]$ ($n \in \mathbb{N}$). On remarque que ces parties sont en particulier alternantes, justifiées, bien ouvertes et à portée stricte. Les stratégies sont celles représentant un entier n, donc définies par :*

$$s(q[\alpha_0]) = r_n\alpha_0[\alpha]$$

ou bien la stratégie non définie :

$$s(q) = \perp.$$

On pourrait considérer uniquement les stratégies bien définies, mais ce jeu sera utilisé pour représenter le type de base de PCF *(dans la section 5.3), qui considère aussi les fonctions partielles (son type de base n'est donc pas $\mathbb{N}$ mais $\mathbb{N} \cup \{\perp\}$). Enfin, on peut considérer un encodage des coups tel que la taille de l'encodage de r_n est de l'ordre de $|n|$ (à une constante près).*

Exemple 5.13. *Le jeu $\mathcal{J}_{\mathbb{N}\to\mathbb{N}}$ est défini à partir du sous-jeu $\mathcal{J}_{\mathbb{N}}$ (exemple 5.12) sur l'arène $\mathcal{A}_{\mathbb{N}\to\mathbb{N}}$ (figure 5.4). Les parties sont toutes les parties alternantes, justifiées, bien ouvertes et à portée stricte, donc préfixes de parties de la forme :*

$$q', q, r_{n_1}, \ldots, q, r_{n_k}, r'_m \tag{5.2}$$

(où les noms et les justifications sont implicites). Autrement dit, une partie est un préfixe d'une partie commençant par une question q', suivie d'une liste de parties dans $\mathcal{J}_{\mathbb{N}}$ (où on a pu effectuer une α-conversion des noms pour garantir la condition d'unicité et que les questions q soient justifiées par la question initiale q') et terminant par une réponse finale r'_m.

La vue *d'une partie terminant par un coup $q\alpha[\beta]$ dans $\mathcal{A}_{\mathbb{N}\to\mathbb{N}}$ est le coup correspondant dans $\mathcal{A}_{\mathbb{N}}$:*

$$\text{vue}(\ldots, q\alpha[\beta]) = q[\alpha_0]$$

la fonction $\text{vue}^{\perp}$ *restaure les noms adéquats dans la réponse de l'opposant :*

$$\text{vue}^{\perp}((\ldots, q\alpha[\beta]), r_n\alpha_0[\gamma]) = r_n\beta[\gamma']$$

où γ' est un nom n'apparaissant pas dans la partie courante. Enfin, de même que dans l'exemple précédent, on peut munir ce jeu d'un encodage des coups pour lequel la taille de l'encodage des réponses r_n et r'_n est de l'ordre de $|n|$.

Exemple 5.14. *De même que précédemment, le jeu $\mathcal{J}_{(\mathbb{N}\to\mathbb{N})\to\mathbb{N}}$ est construit à partir du sous-jeu $\mathcal{J}_{\mathbb{N}\to\mathbb{N}}$, ses parties sont les parties alternantes, justifiées, bien ouvertes et à portée stricte, autrement dit préfixes de parties commençant par q'', suivi d'un nombre fini de parties complètes dans $\mathcal{J}_{\mathbb{N}\to\mathbb{N}}$ (donc de la forme de l'équation 5.2, où les q' sont justifiées par la question initiale q''), terminant par une réponse de la forme r''_n. Cela représente bien le comportement d'une machine à oracle : la machine est initialisée, puis elle interroge son oracle un nombre fini de fois (chaque appel correspondant à une partie dans $\mathcal{J}_{\mathbb{N}\to\mathbb{N}}$), et finit par retourner un résultat et s'arrêter.*

La fonction vue *retourne la sous-partie correspondante :*

$$\mathrm{vue}(\dots q'\alpha_0[\alpha], p) = q'[\alpha_0]p_{\alpha\leftarrow\alpha_0}$$

où p ne contient que des coups héréditairement justifiés par $q'\alpha_0[\alpha]$, et $p_{\alpha\leftarrow\alpha_0}$ est égal à p où α a été remplacé par α_0. Intuitivement, c'est ce que connaît l'oracle à un instant donné, celui-ci n'ayant pas accès aux coups correspondant aux appels précédents. De même que précédemment, la fonction $\mathrm{vue}^{\perp}$ *récrit les noms du coup dans le sous-jeu en noms compatibles avec la partie initiale.*

L'encodage des coups et des noms nous permet d'encoder des parties (enc désignera sans distinction l'encodage sur tous ces ensembles) par des mots binaires. Cela nous permet alors de voir une stratégie comme une fonction d'ordre 1 sur les mots binaires, et on peut alors définir l'ensemble des stratégies calculables.

Définition 5.12 (Stratégie calculable)**.** *Une stratégie s dans un jeu $\mathcal{J}$ est calculable si elle est représentée via* enc *par une fonction calculable. Autrement dit, elle est calculable relativement à* enc*, selon la définition 2.11.*

Dans les exemples utilisés dans ce chapitre, les stratégies représentent des fonctions. On peut alors dire qu'une fonction est calculable si elle est représentée par une stratégie calculable (de même qu'une fonction est Kleene-Kreisel-calculable si elle est représentée par un associé calculable).

Remarque 5.12. *Tout comme en analyse récursive, différents encodages engendrent différentes classes de fonctions calculables. Par exemple, le jeu $\mathcal{J}_{\mathbb{N}\to\mathbb{N}}$ (exemple 5.13) est muni d'un encodage raisonnable, pour lequel les stratégies calculables représentent exactement les fonction calculables de $\mathbb{N}\to\mathbb{N}$. En revanche, si l'on ajoute à la représentation binaire de r_i un bit égal à la valeur d'un problème indécidable en i, alors certaines stratégies calculables pourront représenter des fonction non calculables de $\mathbb{N}\to\mathbb{N}$. De même, deux encodages équivalents en termes de calculabilité pourront engendrer des notions de complexité différentes (voir remarque 5.19).*

Exemple 5.15. *Les jeux $\mathcal{J}_{\mathbb{N}\to\mathbb{N}}$ et $\mathcal{J}'_{\mathbb{N}\to\mathbb{N}}$ définissent les mêmes fonctions calculables. En effet, on peut convertir une stratégie calculable s dans $\mathcal{J}_{\mathbb{N}\to\mathbb{N}}$ en une stratégie calculable s' dans $\mathcal{J}'_{\mathbb{N}\to\mathbb{N}}$ représentant la même fonction :*

$$\forall i\in\mathbb{N}, s'(q_i) = r_j \text{ si } s(q') = r'_j \text{ ou } s(q', q, r_i) = r'_j.$$

Inversement, si s' est calculable dans $\mathcal{J}'_{\mathbb{N}\to\mathbb{N}}$, on pose :

$$\begin{cases} s(q') = q \\ s(q', q, r_i) = r'_j \text{ si } s'(q_i) = r_j. \end{cases}$$

Pour un encodage raisonnable, les stratégies calculables sont exactement celles qui représentent des fonctions calculables.

Exemple 5.16. *De même, les fonctions d'ordre 2 représentées par des stratégies calculables de $\mathcal{J}_{(\mathbb{N}\to\mathbb{N})\to\mathbb{N}}$ sont exactement les fonctions calculables par machines à oracles. En effet, on peut facilement construire une machine calculant une stratégie dans ce jeu à partir d'une machine à oracle et vice-versa.*

Dans la suite, nous ne considérerons plus que des jeux séquentiels effectifs. Pour de tels jeux on peut définir une opération permettant de confronter une stratégie à une stratégie dans un sous-jeu en les faisant jouer alternativement.

Définition 5.13 (Confrontation)**.** *Étant donné une stratégie s dans un jeu séquentiel effectif $\mathcal{J}$ et une stratégie s_i dans un sous-jeu $\mathcal{J}^i$, la* **confrontation** *entre s et s_i est l'opération qui construit une partie p de façon incrémentale ainsi :*

La partie commence avec le coup initial i : $p \leftarrow i[\alpha_0]$. Ensuite, les stratégies s et s_i jouent à tour de rôle comme suit, jusqu'à ce que l'une d'entre elles soit non définie ou que s réponde à la question initiale (i.e. joue un coup de la forme $r\alpha_0[\alpha]$).

La stratégie s joue en ajoutant $s(p)$ à la partie : $p \leftarrow p, s(p)$, puis s_i joue en fonction de la vue du jeu qui lui est donnée ($\mathrm{vue}(p)$). Le coup ainsi obtenu est converti en un coup dans $\mathcal{J}$ et ajouté à la partie : $p \leftarrow p, \mathrm{vue}^{\perp}(p, s_i(\mathrm{vue}(p)))$.

Étant donné s, cette opération définit une fonction partielle de type $\cup_{i\in\mathcal{J}}\mathbb{S}_i \hookrightarrow \mathfrak{F}$: si s termine sa confrontation avec $s_i \in \mathbb{S}_i$ par la réponse finale $r \in \mathfrak{F}$ on notera $s[s_i] = r$ et r est appelé **résultat de la confrontation** *; dans le cas contraire, $s[s_i]$ est indéfini.*

Enfin, on appelle **historique de la confrontation** *l'état final de la partie p, que l'on notera $H(s, s_i)$.*

Exemple 5.17. *Si s_n est la stratégie dans $\mathcal{J}_{\mathbb{N}}$ représentant l'entier n et $s_\perp$ l'unique stratégie de $\mathcal{J}_{\emptyset}$, alors $H(s_n, s_\perp) = q, r_n$ et donc $s_n[s_\perp] = r_n$. Dans le cas où s est la stratégie indéfinie de $\mathcal{J}_{\mathbb{N}}$, alors sa confrontation avec $s_\perp$ ne termine pas et $s[s_\perp]$ est indéfini.*

Exemple 5.18. *La confrontation d'une stratégie sur $\mathcal{J}_{\mathbb{N}\to\mathbb{N}}$ représentant une fonction f avec une stratégie sur $\mathcal{J}_{\mathbb{N}}$ représentant un entier n termine toujours sur la réponse $r'_{f(n)}$. De même, la confrontation entre une stratégie sur $\mathcal{J}_{(\mathbb{N}\to\mathbb{N})\to\mathbb{N}}$ et une stratégie sur $\mathcal{J}_{N\to\mathbb{N}}$ représentant respectivement une fonction F d'ordre 2 et une fonction f d'ordre 1 termine sur $r''_{F(f)}$.*

Remarque 5.13. *Il existe deux raisons pour lesquelles une confrontation peut ne pas terminer. Soit à un tour donné l'une des stratégies est indéfinie, soit elles sont toujours définies, mais la première stratégie ne retourne jamais de réponse finale et la partie se poursuit indéfiniment.*

Exemple 5.19. *La confrontation de la stratégie s dans $\mathcal{J}_{\mathbb{N}\to\mathbb{N}}$ définie pour tout $k \in \mathbb{N}$ par $s(q', q, r_1, \ldots, q, r_k) = q$ face à une stratégie représentant un entier $n \in \mathbb{N}$ ne termine pas car s répète indéfiniment la question q à son opposant.*

Remarque 5.14. *Si on voit une stratégie s comme une fonction calculable d'ordre 1, on peut voir $s[\cdot]$ comme une fonction calculable d'ordre 2. En effet, on peut construire une machine à oracle (schématisée dans la figure 5.8) qui calcule progressivement l'historique de la confrontation en alternant des appels à l'oracle pour obtenir le coup de l'opposant avec des calculs internes pour obtenir la valeur de s sur la partie courante. La machine s'arrête en fournissant la réponse finale de s.*

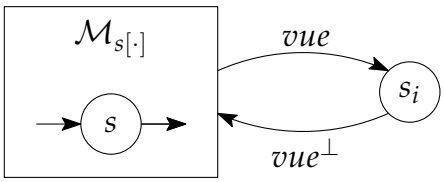

FIGURE 5.8 – Schématisation d'une machine à oracle calculant $s[\cdot]$. Elle peut calculer s et accède aux valeurs d'un opposant s_i via un oracle dont les entrées et sorties sont modifiées par les fonctions $\text{vue}^{\perp}$ et vue.

5.2 Taille et complexité des stratégies

Nous souhaitons maintenant définir une notion de complexité sur les stratégies. Or, la complexité d'un objet est une notion liée à la taille de ses arguments. Si l'on considère la confrontation d'une stratégie avec une stratégie dans une sous-arène, alors pour définir la complexité de la première il faut définir la taille de la seconde. Nous définissons donc la taille des stratégies (sous-section 5.2.1) avant de définir leur complexité de deux manières équivalentes (sous-sections 5.2.2 et 5.2.3). La première approche est basée sur la notion de temps de calcul usuelle d'ordre 1 (puisqu'une stratégie peut être vue comme une fonction d'ordre 1) ; la seconde utilise un modèle de machines plus adapté aux jeux et pour lequel la notion de temps de calcul est plus naturelle. Enfin, nous en déduisons une classe de stratégies calculables en temps polynomial (sous-section 5.2.4).

5.2.1 Taille d'une stratégie

Comme nous l'avons déjà dit, le manque de bonne définition de la taille d'une fonction d'ordre supérieur est l'un des principaux obstacles à une théorie de la complexité d'ordre supérieur. Pour définir la taille d'une stratégie (qui représentera plus tard une fonction d'ordre supérieur), il nous faut déjà définir la taille d'une partie.

C'est en particulier pour cela que nous avons dû spécifier un encodage des parties dans les mots binaires.

Définition 5.14 (Taille d'une partie)**.** *Étant donné un jeu séquentiel effectif $\mathcal{J}$, la taille d'un coup (nommé ou non) est la taille de son encodage binaire, que l'on notera $|c|$. On notera par extension $|p|$ la somme des tailles des coups d'une partie p.*

Remarque 5.15. *On aurait aussi pu définir la taille d'une partie par la taille de sa représentation binaire, autrement dit en prenant en compte la taille des noms. Cela produirait une définition globalement équivalente, la condition sur l'encodage des jeux séquentiels effectifs (définition 5.11) impliquant que chacune de ces définitions borne l'autre à une composition par un polynôme près. Nous choisirons donc la première définition par simplicité. Notons cependant que la taille des noms aura de l'importance pour l'étude des classes de complexité sous-linéaires, comme nous l'avons déjà vu dans la remarque 5.6.*

Dans la suite, on notera $\preccurlyeq$ l'ordre partiel ponctuel défini par induction sur les types finis :

— Si $n, n' \in \mathbb{N}, n \preccurlyeq n' \iff n \leq n'$.

— Si $F, F' : \sigma$ et $G, G' : \tau$, $(F, G) \preccurlyeq (F', G') \iff F \preccurlyeq F' \wedge G \preccurlyeq G'$.

— Si $F, F' : \sigma \to \tau$, $F \preccurlyeq F' \iff \forall f : \sigma, F(f) \preccurlyeq F'(f)$.

On notera de plus $\pi_j : (\prod_{i \in A} \tau_i) \to \tau_j$ la projection d'un type produit sur sa composante j.

Nous pouvons maintenant définir la taille d'une stratégie. Celle-ci sera une fonction dont le type dépend de l'arène dans laquelle elle joue.

Définition 5.15 (Type associé à jeu séquentiel effectif)**.** *Le type fini associé à un jeu séquentiel effectif $\mathcal{J}$ est défini par induction sur la profondeur de son arène.*

— *Si la profondeur de l'arène $\mathcal{A}$ du jeu $\mathcal{J}$ est inférieure ou égale à 1, alors le type $\tau_{\mathcal{J}}$ est égal à $\prod_{i \in \mathfrak{I}} \mathbb{N} = \mathbb{N}^{\#\mathfrak{I}}$,*

— *sinon, $\tau_{\mathcal{J}}$ est défini à partir des types associés à chacun de ses sous-jeux ainsi :*

$$\tau_{\mathcal{J}} = \prod_{i \in \mathfrak{I}} (\tau_{\mathcal{J}^i} \to \mathbb{N})$$

où $\mathfrak{I}$ est l'ensemble des questions initiales de $\mathcal{J}$.

Remarque 5.16. *Les types ainsi définis sont bien des types finis. En particulier, chacun des produits est fini, car le nombre de coups initiaux d'un jeu séquentiel effectif est fini.*

Exemple 5.20. *Comme on s'y attend, les types associés aux jeux $\mathcal{J}_{\mathbb{N}}$, $\mathcal{J}_{\mathbb{N}\to\mathbb{N}}$ et $\mathcal{J}_{(\mathbb{N}\to\mathbb{N})\to\mathbb{N}}$ sont respectivement $\mathbb{N}$, $\mathbb{N} \to \mathbb{N}$ et $(\mathbb{N} \to \mathbb{N}) \to \mathbb{N}$. Le type associé au jeu vide est le type particulier* unit *à un seul élément.*

On peut maintenant définir la taille d'une stratégie dans $\mathcal{J}$ comme une fonction de type $\tau_{\mathcal{J}}$. Elle est définie comme la taille maximale de l'historique d'une confrontation face à des stratégie de taille bornée (dans un sous-jeu), étant donnée une telle borne.

Définition 5.16 (Taille d'une stratégie)**.** *La taille d'une stratégie s dans une jeu $\mathcal{J}$ sur une arène $\mathcal{A}$ est définie par induction sur $\mathcal{J}$ (ou de façon équivalente, sur la profondeur de $\mathcal{A}$).*

— *Si la profondeur de l'arène est d'au plus 1, alors pour tout coup initial i, la sous-arène $\mathcal{A}^i$ n'a pas de coups du joueur, donc $\mathcal{J}^i$ possède au plus une stratégie, $\bot_i$ définie nulle part. On définit alors la taille $\|s\|_{\mathcal{J}} : \tau_{\mathcal{J}}$ de la stratégie s comme le produit cartésien sur $i \in \mathfrak{I}$ de la taille de son historique contre $\bot_i$:*

$$\forall s \in \mathbb{S}, \|s\|_{\mathcal{J}} = (|H(s, \bot_i)|)_{i \in \mathfrak{I}}.$$

— *Si la profondeur de $\mathcal{A}$ est au moins 2, alors la fonction $\|\cdot\|_{\mathcal{J}} : \tau_{\mathcal{J}}$ est définie par :*

$$\forall s \in \mathbb{S}, \|s\|_{\mathcal{J}} = \left(\lambda b. \sup_{\substack{s_i \in \mathbb{S}_i \\ \|s_i\|_{\mathcal{J}^i} \preccurlyeq b}} |H(s, s_i)| \right)_{i \in \mathfrak{I}}$$

En particulier, pour les jeux définis en exemple, cette notion de taille fait sens. En effet, la taille d'une stratégie représentant un entier est de l'ordre de la taille de la représentation binaire de cet entier.

Proposition 5.1. *Dans le jeu $\mathcal{J}_{\mathbb{N}}$, tout entier n a une unique stratégie de taille bornée par $c_0 \cdot |n|$, pour une constante c_0. Inversement, une stratégie de taille k représente un entier de taille inférieure à k.*

Preuve

La confrontation d'une stratégie représentant un entier n face à la stratégie $\bot_q$ conduit à l'historique q, r_n, dont la taille est bien de l'ordre de la taille de la représentation binaire de n. Le second point est immédiat, puisque la taille de l'encodage de r_n est plus grand que $|n|$. □

De même toute fonction f d'ordre 1 a une stratégie dont la taille est approximativement la taille de f, *i.e.* $|f|_1$, telle que définie par Kapron et Cook (définition 1.8).

Proposition 5.2. *Toute fonction $f : \mathbb{N} \to \mathbb{N}$ a une stratégie dont la taille est bornée par $\lambda k.c + k + |f|_1(k)$ pour une constante c indépendante de f.*

Inversement, $|f|_1 \preccurlyeq \lambda k.\|s_f\|_{\mathcal{J}_{\mathbb{N}\to\mathbb{N}}}(c_0 \cdot k)$ pour toute stratégie s_f représentant f.

Preuve

On considère la stratégie canonique qui demande son argument x, puis répond la valeur de f sur x (cf. exemple 5.8). La confrontation face à une stratégie dans $\mathcal{J}_{\mathbb{N}}$ de taille inférieure à k (donc représentant un entier x de taille inférieure à k d'après la proposition 5.1) produit un historique de la forme $q', q, r_x, r'_{f(x)}$. Comme la taille de l'encodage de $r'_{f(x)}$ est de l'ordre de la taille binaire de x, la taille de cette partie est bornée par $c + |x| + |f(x)|$ pour une certaine constante c, ce qui implique le premier résultat en maximisant sur toutes les stratégies de taille inférieure à k.

Pour le second point, il suffit de constater que la propriété 5.1 implique que $\{s \mid \|s\|_{\mathcal{J}_{\mathbb{N}}} \leq c_0 \cdot k\}$ contient toutes les stratégies calculant des entiers de taille inférieure à k. Leur historique

face à s_f contient alors $r'_{f(n)}$ pour de tels entiers n. Comme $|r'_{f(n)}| \geq |f(n)|$, on a $\|s_f\|_{\mathcal{J}_{\mathbb{N}\to\mathbb{N}}}(c_0 \cdot k) \geq \max_{|n|\leq k} |f(n)| = |f|_1(k)$. □

Proposition 5.3. *Une fonction de type $\mathbb{N} \to \mathbb{N}$ a une stratégie de taille bornée si et seulement si elle est totale.*

Preuve

Ce résultat est immédiat, étant donné que si la taille d'une stratégie est bornée, c'est que sa confrontation termine face à toute stratégie dans $\mathcal{J}_{\mathbb{N}}$ de taille bornée, autrement dit, toute stratégie représentant un entier. □

Il n'existait pas jusqu'ici de mesure intéressante de la taille d'une fonction d'ordre 2. Nous avons vu, dans le cadre de l'analyse récursive, que la taille de la représentation d'une fonction réelle était liée à son module de continuité (cf. proposition 2.5). On peut de même définir le module de continuité d'une fonction de type $(\mathbb{N} \to \mathbb{N}) \to \mathbb{N}$ et le lier à la taille des stratégies la représentant.

Définition 5.17. *Une fonction $B : (\mathbb{N} \to \mathbb{N}) \to \mathbb{N}$ est un module de continuité pour une fonction $F : (\mathbb{N} \to \mathbb{N}) \to \mathbb{N}$ si :*

$$\forall b, f, g : \mathbb{N} \to \mathbb{N}, (f, g \preccurlyeq b \wedge \forall n \leq B(b), f(n) = g(n)) \implies F(f) = F(g).$$

Autrement dit, un module de continuité donne une borne supérieure sur la quantité d'information de l'entrée dont dépend la fonction.

Proposition 5.4. *Si $F : (\mathbb{N} \to \mathbb{N}) \to \mathbb{N}$ est représentée par une stratégie (dans $\mathcal{J}_{(\mathbb{N}\to\mathbb{N})\to\mathbb{N}}$) de taille $T : (\mathbb{N} \to \mathbb{N}) \to \mathbb{N}$, alors $\lambda b.2^{T(\lambda k.c+k+|b|_1(k))}$ est un module de continuité pour F (où c est une constante indépendante de F).*

Preuve

Soient $f, g, b : \mathbb{N} \to \mathbb{N}$ tels que f et g sont bornés par b et leurs valeurs sont égales sur tous les entiers inférieurs à $2^{T(\lambda k.c+k+|b|_1(k))}$. D'après la proposition 5.2, f et g sont représentées par des stratégies de taille $\lambda k.c + k + |b|_1(k)$. L'historique de la confrontation de la stratégie de taille T représentant F face à ces deux stratégies est donc de taille inférieure à $T(\lambda k.c + k + |b|_1(k))$ par définition de la taille d'une stratégie. Cet historique ne peut donc pas contenir de réponse de taille plus grande que $T(\lambda k.c + k + |b|_1(k))$, autrement dit, la stratégie calculant F n'a pas pu évaluer les valeurs de f et g pour des entrées de taille supérieure à cette borne. Comme elle ne peut différencier ces deux stratégies, les deux historiques sont égaux et en particulier le résultat de la confrontation est identique, ce qui revient à dire que $F(f) = F(g)$. □

Cette proposition illustre en particulier le fait que la taille d'une stratégie mesure la quantité d'information utilisée par la stratégie avant de fournir une réponse finale.

Inversement, les fonctions possédant un module de continuité sont représentables par une stratégie dont la taille est liée à ce module.

Proposition 5.5. *Si $F : (\mathbb{N} \to \mathbb{N}) \to \mathbb{N}$ a un module de continuité B, alors elle est représentée par une stratégie dont la taille est bornée.*

Preuve

Soit F une telle fonction, qui est donc continue. On peut alors définir une stratégie s pour F sur le principe de l'associé canonique défini par l'équation (4.1). Celle-ci demande les valeurs successives de son entrée f en 0, 1, ... n jusqu'à ce que F soit constante sur l'ensemble des fonctions ayant les mêmes valeurs sur $\{0, 1, \ldots, n\}$, puis retourne la valeur de cette constante. Soit s_f une stratégie calculant une fonction $f : \mathbb{N} \to \mathbb{N}$. L'historique de s face à s_f est composé de n évaluations de f, chacune étant de taille inférieure à $\|s_f\|(c_0 \cdot |n|)$ d'après la proposition 5.1. De plus, n est lui-même borné par $B(|f|_1)$ par définition, qui peut à son tour être borné en fonction de $\|s_f\|$ d'après la deuxième partie de la proposition 5.2. Enfin, la taille de la réponse finale (correspondant à $F(f)$) est bornée (environ) par $\max_{|g|_1 \leq |f|_1} F(g)$, qui peut aussi s'exprimer en fonction de $\|s_f\|$. La taille de l'historique de la confrontation est donc bien borné en fonction de la taille de s_f, ce qui nous permet de conclure. □

Ces deux propositions réunies permettent de caractériser les fonctions d'ordre 2 représentables par des stratégies de taille finie.

Théorème 5.1. *Une fonction de* $(\mathbb{N} \to \mathbb{N}) \to \mathbb{N}$ *est représentée par une stratégie de* $\mathcal{J}_{(\mathbb{N}\to\mathbb{N})\to\mathbb{N}}$ *de taille bien définie si et seulement si elle est continue.*

Remarque 5.17. *Vues comme des fonctions, les stratégies de taille bornée forment une classe de fonctions héréditairement totales. En effet, si la taille de s est bornée, alors pour toute stratégie* s' *de taille bornée, la confrontation entre s et* s' *termine, et en particulier, on a* $|s[s']| \leq \|s\|(\|s'\|)$.

5.2.2 Complexité cumulée

La taille des entrées est un point important dans les définitions usuelles de complexité. Maintenant que nous avons une telle notion sur les stratégies, nous allons en déduire une première définition de complexité. La définition la plus immédiate de la complexité d'une stratégie est celle de la complexité de la fonction binaire correspondante (cf. définition 5.12). De la même façon que l'on a défini qu'une stratégie est calculable si la fonction d'ordre 1 correspondante (*i.e.* son représentant relativement à enc) est calculable (définition 5.12), on pourrait dire qu'elle est calculable en temps $t : \mathbb{N} \to \mathbb{N}$ si la fonction correspondante l'est aussi. Cependant, cette définition souffre du même problème que celui soulevé par les associés de Kleene (illustré par l'exemple 4.1) puisqu'elle ne prend pas en compte la longueur du dialogue entre la stratégie et son opposant. Autrement dit, on peut souvent construire une stratégie de faible complexité qui pose des questions inutiles pour augmenter arbitrairement la longueur de l'historique.

Une autre solution est d'essayer de mesurer le temps total nécessaire à simuler la confrontation. Il faut alors additionner le temps nécessaire pour jouer chaque coup de l'historique.

Définition 5.18 (Complexité cumulée)**.** *Une stratégie s d'un jeu* $\mathcal{J}$ *est calculable en temps cumulé* $T : \tau_{\mathcal{J}}$*, si s est calculable par une machine de Turing (au sens de la définition 5.12) de*

temps de calcul T_0, et pour toute stratégie s_i dans le jeu $\mathcal{J}^i$, si

$$H(s, s_i) = c_1, c'_1, \dots c_n, c'_n,$$

alors

$$\sum_{1 \leq j \leq n} |c_j| + T_0(c_1, c'_1, \dots, c_j) \leq \pi_i(T(\|s_i\|_{\mathcal{J}^i})). \tag{5.3}$$

Cela revient à dire que le temps de calcul de la machine à oracle calculant $s[]$ (décrite dans la remarque 5.14) sur l'oracle s_i (ou plutôt sa représentation par enc*) est borné par $T(\|s_i\|_{\mathcal{J}^i})$.*

L'intuition derrière cette définition est de borner le temps nécessaire à la confrontation de la stratégie face à une stratégie de taille donnée. Celui-ci est mesuré par la taille des coups joués par l'opposant (c'est le coût d'un appel dans une machine à oracle) ainsi que par le temps nécessaire à la stratégie pour jouer chacun de ses coups c'_i en temps $T_0(c_1, c'_1, \dots c_i)$.

Remarque 5.18. *On aurait pu inclure le temps de calcul des fonctions* vue *et* vue$^\perp$ *dans la complexité cumulée, cependant nous avons fait l'hypothèse que leur complexité est polynomiale, cette définition serait donc polynomialement équivalente à la précédente.*

Pour mieux comprendre cette définition, appliquons-la aux fonctions usuelles. Tout d'abord, dans le cas des entiers, la complexité est de l'ordre de la taille.

Exemple 5.21. *D'après la proposition 5.1, tout entier $n \in \mathbb{N}$ a une unique stratégie s dans $\mathcal{J}_{\mathbb{N}}$ de taille de l'ordre de $|n|$. Sa complexité est aussi de l'ordre de $|n|$. En effet, $H(s, \perp) = q, r_n$, et $|q| + T_0(q) = \mathcal{O}(|n|)$, où T_0 est la complexité d'une machine constante calculant un encodage de r_n.*

Le seul autre cas de stratégie dans $\mathcal{J}_{\mathbb{N}}$ est celui de la stratégie indéfinie, dont la taille n'est pas bornée car la partie correspondante ne termine pas. A fortiori, sa complexité ne l'est pas non plus.

Exemple 5.22. *De même, si f est une fonction de type $\mathbb{N} \to \mathbb{N}$ calculable par une machine de Turing en temps T, d'après la propriété 5.2, elle est représentée par une stratégie $s_f \in \mathbb{S}_{\mathcal{J}_{\mathbb{N} \to \mathbb{N}}}$ dont la taille est bornée par $\lambda k.c + k + |f|_1(k)$. La complexité cumulée de cette même stratégie est alors bornée par une fonction de la forme $\lambda k.c' + k + T(k)$. En effet, on peut facilement adapter une telle machine calculant f en une machine calculant s_f ayant la même complexité. De plus, une stratégie s' dans $\mathcal{J}_{\mathbb{N}}$ de taille inférieure à k représente un entier n de taille inférieure à k. On a alors $H(s_f, s') = q', q, r_n, r_{f(n)}$, et*

$$|q'| + T(q') + |q, q', r_n| + T(q, q', r_n) = \mathcal{O}(|n| + T(|n| + c)).$$

Inversement, si une stratégie calculable en temps T représente une fonction f de $\mathbb{N} \to \mathbb{N}$, alors f est calculable par une machine de Turing en temps T (à une composition polynomiale près).

Cette définition de complexité est donc polynomialement équivalente à la définition usuelle, ce qui implique en particulier la proposition suivante.

Proposition 5.6. *Une fonction de $\mathbb{N} \to \mathbb{N}$ est représentée par une stratégie de $\mathcal{J}_{\mathbb{N}\to\mathbb{N}}$ de complexité cumulée polynomiale si et seulement si elle est elle-même calculable en temps polynomial.*

Remarque 5.19. *Le choix d'un encodage change de façon importante la notion de complexité induite, par exemple si l'on choisit un encodage des réponse r_n et r'_n de taille n (et non plus $|n|$).*

Exemple 5.23. *Si $F : (\mathbb{N} \to \mathbb{N}) \to \mathbb{N}$ est calculable en temps T par une machine à oracle $\mathcal{M}$, alors la complexité cumulée de la stratégie s_F définie à partir de $\mathcal{M}$ dans l'exemple 5.16 est bornée polynomialement en T. En effet, si s_f est une stratégie de $\mathcal{J}_{\mathbb{N}\to\mathbb{N}}$ représentant une fonction $f : \mathbb{N} \to \mathbb{N}$, alors l'historique $H(s_F, s_f)$ est de la forme $q'', p_1, \dots p_k, r''_v$, où chaque p_i correspond (à un renommage près) à une partie de s_f face à une stratégie représentant un entier n_i (qui correspond à un appel à l'oracle sur l'entrée n_i). La taille de chacune de ces parties est de l'ordre de $|f|_1(|n_i|)$ dont la somme est bornée par $T(|f|_1)$. Donc la somme des coups de l'opposant dans l'historique est bornée polynomialement par $T(|f|_1)$. De plus le temps de calcul de la machine calculant s_F est aussi bornée polynomialement par $T(|f|_1)$ sur chaque préfixe de l'historique, donc la complexité cumulée est elle aussi bornée polynomialement en $T(|f|_1)$.*

Inversement, si s_F est une stratégie dans $\mathcal{J}_{(\mathbb{N}\to\mathbb{N})\to\mathbb{N}}$ de complexité cumulée T, alors on peut construire une machine à oracle à partir de la machine calculant s_F, chaque sous-partie correspondant à un appel à l'oracle. La complexité de cette machine est alors aussi de l'ordre de T.

On en déduit la même équivalence avec $\textsc{fptime}_2$.

Proposition 5.7. *Une fonction de $(\mathbb{N} \to \mathbb{N}) \to \mathbb{N}$ est représentée par une stratégie de $\mathcal{J}_{(\mathbb{N}\to\mathbb{N})\to\mathbb{N}}$ de complexité cumulée polynomiale (d'ordre 2) si et seulement si elle est elle-même calculable en temps polynomial par une machine à oracle.*

Un argument permettant de justifier cette définition est que la complexité d'une stratégie borne sa taille. C'était déjà le cas, par exemple pour les fonctions d'ordre 1 (cf. remarque 1.6), ou pour les fonctions de $\mathcal{C}[0,1]$ (cf. théorème 2.3).

Proposition 5.8. *La complexité cumulée d'une stratégie borne sa taille.*

Preuve

C'est immédiat d'après la définition de la complexité cumulée. En effet, celle-ci borne la somme de la taille des coups de l'opposant, ainsi que la somme des temps de calculs pour calculer les coups du joueur. Le temps de calcul d'un coup (ou plutôt de son encodage) est plus grand que sa taille, donc la somme totale est plus grande que la taille de l'historique, ce qui prouve le résultat. □

Remarque 5.20. *Comme nous l'avions prévu, cette proposition (ainsi que la remarque 5.17) implique que seules les stratégies définies (au sens où la confrontation termine) sur toutes les*

stratégies de taille bornée peuvent avoir une complexité bornée. Nous avons déjà rencontré ce phénomène (i.e. le fait que la calculabilité et la complexité ne dépendent pas seulement de la capacité de calcul, mais aussi de la taille) dans le cas des fonctions réelles, où une fonction est calculable (ou calculable en temps polynomial) si et seulement si son approximation sur $\mathbb{Q}$ *l'est (cf. définition 2.17)* et *si son module de continuité est borné (respectivement, borné par un polynôme).*

On peut définir la complexité cumulée d'une stratégie relativement à un oracle, de manière similaire à la complexité d'appel à l'ordre 2 (définition 1.10) et la relier à sa taille. C'est en effet ce que nous avons déjà vu avec les fonctions réelles (cf. théorème 2.3) où le rôle de la taille était joué par le module de continuité (qui est proche de la notion de taille pour une stratégie, d'après la proposition 5.4). Nous avons même défini (en quelque sorte) des notions de taille (déterministe et non-déterministe) sur les normes réelles pour pouvoir caractériser leur complexité d'appel (cf. théorèmes 3.6 et 3.8).

On définit la ***complexité cumulée relativisée*** d'une stratégie à partir de la définition 5.18 en demandant à ce que s soit calculable en temps T_0 *relativement à un oracle*. On remarque alors que cela revient à définir sa taille.

Proposition 5.9. *Une stratégie a une complexité relativisée bornée par T si et seulement si sa taille est bornée par T.*

Preuve

La propriété 5.8 est aussi valable si le calcul est relativisé, ce qui fournit la première implication.

Si $f_s : \Sigma^* \to \Sigma^*$ est la fonction associant l'encodage d'une partie à l'encodage du coup joué par s sur celle-ci, alors f_s est calculable en temps $|f_s|_1$ relativement à un oracle (cet oracle étant précisément f_s). Le membre de gauche de l'équation 5.3 est alors précisément égal à la taille de l'historique, autrement dit, la complexité cumulée relativisée est bornée par la taille de s. □

5.2.3 Machines à jeux

La complexité cumulée semble correspondre aux notions usuelles, en particulier pour les classes FPTIME et FPTIME_2 d'après les propositions 5.6 et 5.7. Cependant, cette notion est relativement peu naturelle et difficile à manipuler. Nous allons donc définir un modèle de machines proche des machines à oracles usuelles, plus adapté aux jeux et à l'idée sous-jacente au processus de confrontation. La notion de complexité induite sera polynomialement équivalente à la précédente ce qui appuiera sa pertinence et sa robustesse.

Définition 5.19 (Machine à jeux)**.** *Étant donné un jeu $\mathcal{J}$, une machine sur $\mathcal{J}$ est une machine à oracle avec un état initial pour chaque question initiale. Si la machine est dans l'état initial correspondant à la question initiale i, alors son oracle se comportera comme une*

stratégie dans $\mathcal{J}^i$ dans le sens suivant. Lorsque la machine effectue un appel à l'oracle, son ruban spécial contient l'encodage d'un coup c (on dit alors que la machine joue *le coup c). L'oracle répond alors en écrivant l'encodage d'un coup c'. Si l'oracle représente une stratégie s_i, alors $c' = \mathrm{vue}^{\perp}(p, s_i(\mathrm{vue}(p)))$ où p est la partie correspondant aux coups joués par la machine et l'oracle depuis le début de l'exécution. Autrement dit, un appel à l'oracle simule une étape de la confrontation entre une stratégie sur $\mathcal{J}$ (représentée par la machine) et une stratégie sur $\mathcal{J}^i$ (représente par l'oracle). Le jeu s'arrête lorsque la machine entre dans son état final. Le ruban de réponse encode alors un coup final pour $\mathcal{J}$.*

En termes de calculabilité, ce modèle n'est pas différent de celui interprétant les stratégies comme des fonctions d'ordre 1.

Proposition 5.10. *Une stratégie est calculable en tant que fonction d'ordre 1 si et seulement si elle est calculable par une machine à jeux.*

Preuve

Étant donné une machine à jeux calculant une stratégie, on peut définir une machine de Turing standard la calculant. Celle-ci, étant donné l'encodage d'une partie en entrée, simule la machine initiale en répondant à ses appels à l'oracle par les coups successifs de l'opposant dans la partie d'entrée. Lorsque toute la partie a été jouée (autrement dit lorsqu'il n'y a plus de coups pour simuler une réponse de l'opposant), son prochain appel à l'oracle est retourné par la machine (*i.e.* elle retourne l'encodage du coup que la machine initiale aurait joué).

Inversement, on peut construire une machine à jeux qui retient la partie courante et, à chaque fois qu'elle doit jouer, simule la machine de Turing sur celle-ci. Autrement dit, cela correspond à la machine calculant l'opérateur de confrontation (cf. remarque 5.14) dans laquelle on a remplacé le premier oracle par la machine calculant la première stratégie. □

Remarque 5.21. *Comme noté dans la preuve précédente, une machine à jeux peut être vue comme une machine simulant une confrontation face à une stratégie fournie en oracle.*

Ce modèle nous permet maintenant de définir une notion de complexité sur les stratégies de manière similaire à celle des machines à oracles.

Définition 5.20 (Complexité d'une stratégie)**.** *Le* **temps de calcul** *d'une machine à jeux est le même que pour une machine à oracle : le coût d'un appel est la taille du coup correspondant.*

On dit alors qu'une machine sur $\mathcal{J}$ a une **complexité** *$T : \tau_{\mathcal{J}}$ si pour toute stratégie s_i dans $\mathcal{J}^i$ de taille bornée, le temps de calcul de la machine sur l'oracle s_i est borné par $\pi_i(T)(\|s_i\|_{\mathcal{J}^i})$.*

À partir de maintenant, on dira que la **complexité d'une stratégie** *est bornée par T s'il existe une machine à jeux de complexité T la calculant.*

De même que pour la complexité cumulée, la complexité d'une stratégie borne sa taille.

Proposition 5.11. *La complexité d'une stratégie borne sa taille.*

Preuve

Le temps de calcul d'une machine à jeux inclut le temps d'écrire les coups du joueur sur le ruban d'appel, et le coût des coups joués par l'oracle, il borne donc la taille de l'historique correspondant. La complexité et la taille étant tous les deux des suprema sur la taille de l'entrée, la première borne alors bien la seconde. □

Corollaire 5.1. *Le temps de calcul d'une machine à jeux sur une stratégie donnée borne la taille et donc aussi la longueur (autrement dit le nombre de coups) de la partie correspondante.*

Nous allons maintenant montrer que cette définition de complexité et la précédente sont équivalentes.

Proposition 5.12. *La complexité cumulée d'une stratégie borne sa complexité. Inversement, sa complexité cumulée est bornée polynomialement (quadratiquement) en fonction de sa complexité.*

Preuve

Cette preuve réutilise les constructions de la proposition 5.10. Premièrement, si s est calculée par une machine de Turing en temps T, alors le temps de calcul de la machine à jeu construite à partir de celle ci est borné de la façon suivante. Elle simule la machine initiale sur la partie courante en temps $T(p)$ puis attend la réponse de l'oracle en temps $|c|$, où c est la réponse de l'oracle. La somme de ces termes est exactement la complexité cumulée, celle-ci borne donc la complexité de la machine à jeux.

Inversement, si s est calculée par une machine à jeux de complexité T, alors pour toute stratégie d'entrée s_i telle que $H(s, s_i) = c_1, c'_1, \dots c_n, c'_n$, la complexité T' de la machine de Turing ainsi construite vérifie :

$$\forall 1 \leq k \leq n, T'(|c_1, c'_1, \dots c_k|) \leq T(\|s_i\|_{\mathcal{J}^i}) \wedge |c_k| \leq T(\|s_i\|_{\mathcal{J}^i}).$$

En sommant ces inégalités, on obtient :

$$\sum_{1 \leq k \leq n} |c_k| + T'(|c_1, c'_1, \dots c_k|) \leq 2n \cdot T(\|s_i\|_{\mathcal{J}^i}).$$

D'après le corollaire 5.1, le nombre $2 \cdot n$ de coups est lui-même borné par $T(\|s_i\|_{\mathcal{J}^i})$. En généralisant sur s_i, on peut donc conclure que la complexité cumulée est bornée par T^2. □

Le fait que la complexité cumulée borne la complexité n'est pas surprenant. En effet, la complexité cumulée est une majoration grossière du « temps de calcul » nécessaire à une confrontation : une stratégie a besoin de répéter les mêmes calculs plusieurs fois, sur des préfixes croissants de l'historique (de la même façon qu'un associé de Kleene répond 0 s'il n'a pas assez d'information et doit alors recommencer son calcul sur un préfixe plus grand d'un associé en entrée) et est donc inefficace en ce sens. Au contraire, une machine à jeux décrit un processus plus interactif où les calculs sont effectués au fur et à mesure des appels. Malgré cela, ces deux définitions sont polynomialement équivalentes, ce qui les rend davantage robustes.

5.2.4 Temps polynomial

Comme la taille des entrées des machines que nous venons de définir sont d'ordre quelconque, pour définir une classe de complexité nous avons besoin d'une classe de fonctions d'ordre supérieur. Les polynômes usuels ont d'abord été étendus à l'ordre 2 par Kapron et Cook [KC96] pour définir $\textsc{fptime}_2$, puis étendus sur tous les types finis [IRK02] pour caractériser BFF.

Définition 5.21 (Polynômes d'ordre supérieur)**.** *Les polynômes d'ordre supérieur sont les termes du lambda-calcul simplement typé sur* $\mathbb{N}$ *muni des constantes pour l'addition et la multiplication.*

Ces fonctions sont stables par substitution, λ-abstraction et correspondent aux polynômes usuels aux ordre 1 et 2.

Proposition 5.13. *Les polynômes d'ordre supérieur d'ordre 1 sont les polynômes usuels, et ceux d'ordre 2 sont les polynômes d'ordre 2 (définition 1.11).*

On peut maintenant définir naturellement les stratégies calculables en temps polynomial.

Définition 5.22. *Une stratégie est calculable en temps polynomial si et seulement si il existe une machine à jeux la calculant telle que son temps de calcul est borné par un polynôme d'ordre supérieur.*

On peut, de même que dans le cas de la complexité cumulée, définir la complexité relativisée des machine à jeux. Les propositions 5.9 et 5.12 impliquent donc le corollaire suivant faisant le lien entre complexité et taille.

Corollaire 5.2. *Une stratégie est calculable par une machine à jeux en temps polynomial relativement à un oracle si et seulement si sa taille est bornée par un polynôme d'ordre supérieur.*

5.3 Complexité des fonctions d'ordre supérieur

Dans cette section nous appliquons les définition de complexité aux stratégies innocentes (esquissées dans la section 4.5) et en déduirons une notion de complexité pour un sous-ensemble de PCF (en effet, comme nous l'avons déjà dit, il est difficile de définir une notion de complexité sur toutes les fonctions partielles), ainsi qu'une classe de fonctions calculables en temps polynomial. Nous comparerons cette classe avec les classes connues et montrerons que c'est un bon candidat pour un analogue de FPTIME à tout ordre.

5.3.1 Arènes pour PCF

Les types de PCF sont construit à partir des types $\mathbb{N}$ des entiers, $\mathbb{B}$ des booléens et des constructeurs $\to$ et $\times$. Il s'agit donc de définir les arènes correspondant à chacun de ces types. Nous avons déjà défini l'arène $\mathcal{A}_{\mathbb{N}}$ dans l'exemple 5.2. L'arène $\mathcal{A}_{\mathbb{B}}$ est trivialement définie comme une restriction de l'arène $\mathcal{A}_{\mathbb{N}}$ en supprimant les réponses r_n pour $n \geq 2$.

On construit donc par induction l'arène $\mathcal{A}_\tau$ pour tout type fini τ.

Définition 5.23 (Produit d'arènes)**.** *L'arène produit $\mathcal{A}_{\sigma\times\tau}$ est définie simplement comme l'union disjointe des arènes $\mathcal{A}_\sigma$ et $\mathcal{A}_\tau$ (où l'on a implicitement renommé les coups pour assurer la disjonction) :*

$$\mathcal{A}_{\sigma\times\tau} = (\mathfrak{Q}_\sigma \sqcup \mathfrak{Q}_\tau, \mathfrak{R}_\sigma \sqcup \mathfrak{R}_\tau, \lambda_\sigma \amalg \lambda_\tau, \vdash_\sigma \sqcup \vdash_\tau, \mathfrak{I}_\sigma \sqcup \mathfrak{I}_\tau).$$

Exemple 5.24. *L'arène $\mathcal{A}_{\mathbb{N}\times\mathbb{N}}$ est l'union de deux copies disjointes de $\mathcal{A}_{\mathbb{N}}$. Intuitivement, jouer la question initiale de la première (respectivement, de la seconde) copie permet d'évaluer le premier (respectivement, le second) élément d'une paire.*

Définition 5.24 (Exponentiation d'arènes)**.** *L'arène $\mathcal{A}_{\sigma\to\tau}$ est elle-aussi construite à partir de l'union disjointe des arènes $\mathcal{A}_\sigma$ et $\mathcal{A}_\tau$, mais seules les questions initiales de $\mathcal{A}_\tau$ le restent, et autorisent de plus les questions qui étaient initiales dans $\mathcal{A}_\sigma$. De plus, la polarité des coups de $\mathcal{A}_\sigma$ est inversée. Plus formellement :*

$$\mathcal{A}_{\sigma\to\tau} = (\mathfrak{Q}_\sigma \sqcup \mathfrak{Q}_\tau, \mathfrak{R}_\sigma \sqcup \mathfrak{R}_\tau, \lambda_\sigma^\perp \amalg \lambda_\tau, \vdash_\sigma \cup \vdash_\tau \cup(\mathfrak{I}_\tau \times \mathfrak{I}_\sigma), \mathfrak{I}_\tau).$$

Cette construction est illustrée par la figure 5.9, où $\mathcal{A}^\perp$ représente l'arène $\mathcal{A}$ dans laquelle la polarité des coups a été inversée.

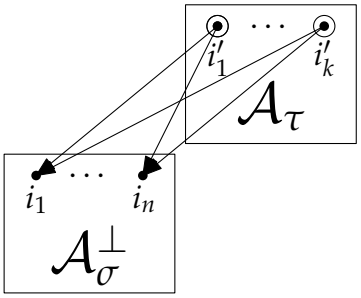

FIGURE 5.9 – Représentation graphique de la construction de l'arène $\mathcal{A}_{\sigma\to\tau}$ à partir des arènes $\mathcal{A}_\sigma$ et $\mathcal{A}_\tau$.

Exemple 5.25. *L'arène $\mathcal{A}_{\mathbb{N}\to\mathbb{N}}$ est bien égale (comme sa notation l'indique) à l'exponentiation de l'arène $\mathcal{A}_{\mathbb{N}}$ avec elle-même. De même , l'arène $\mathcal{A}_{(\mathbb{N}\to\mathbb{N})\to\mathbb{N}}$ est l'exponentiation de $\mathcal{A}_{\mathbb{N}\to\mathbb{N}}$ vers $\mathcal{A}_{\mathbb{N}}$.*

Les arènes ainsi construites forment des forêts finies dont le nombre de racines indique le nombre de composantes dans le type produit, et la profondeur l'ordre du type correspondant. Il est en effet facile de vérifier que le type correspondant à l'arène $\mathcal{A}_\tau$ (définition 5.15) est bien égal à τ.

Remarque 5.22. *Comme on peut s'y attendre, les arènes curryfiées et décurryfiées sont égales. Plutôt qu'une preuve formelle, on peut se contenter de la preuve visuelle fournie par la figure 5.10.*

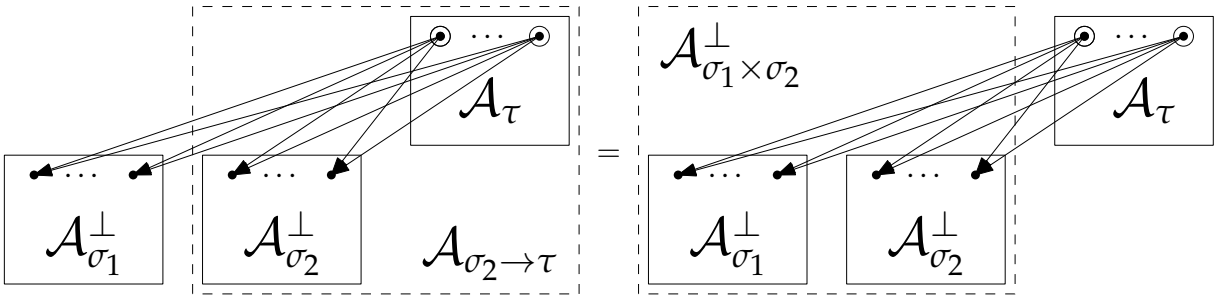

FIGURE 5.10 – Illustration de l'identité entre les arènes $\mathcal{A}_{\sigma_1\to(\sigma_2\to\tau)}$ (à gauche) et $\mathcal{A}_{(\sigma_1\times\sigma_2)\to\tau}$ (à droite).

On peut alors voir tout jeu comme un jeu sur la forme normale de son type associé, i.e. de la forme $(\prod_i \sigma_i) \to \mathbb{N}$ (où les σ_i sont eux-mêmes sous forme normale) ou un produit de types de cette forme. Cette forme est en particulier utile puisqu'elle met en avant le type associé à la sous-arène (ici $\prod_i \sigma_i$).

Remarque 5.23. *On peut de plus définir un encodage canonique des coups dans une telle arène via un parcours en largeur de telles forêts (en commençant par les coups initiaux). La taille de l'encodage d'une question est alors bornée par une constante (dépendant du type) et la taille d'une réponse représentant un entier n est bornée par la taille de la représentation binaire de n à une constante dépendant du type près.*

5.3.2 Jeux innocents

Les jeux innocents sont les jeux utilisés pour obtenir une sémantique de PCF (indépendamment définis par Nickau [Nic94 ; Nic+96] et Hyland et Ong [HO00]). Ils sont ici présentés selon notre approche.

Définition 5.25 (Jeux innocents)**.** *Étant donné un type fini τ, le jeu $\mathcal{J}_\tau$ est défini sur l'arène $\mathcal{A}_\tau$. Ses parties sont alternantes, bien-ouvertes, bien parenthésées, à portée stricte, et toute réponse justifie nécessairement la dernière question ouverte. On utilise l'encodage des coups décrit dans la remarque 5.23. Enfin, la vue d'une partie p est la vue innocente définie ainsi :*

- *si une question ouverte q est justifiée par une question q, alors tous les coups entre q' et q sont enlevés de p ;*
- *si r réponds à une question q, alors tous les coups entre q et r sont enlevés de p ;*

— *les noms des coups sont modifiés pour obtenir une partie valide (en particulier le coup initial $i\alpha[\beta]$ dans $\mathrm{vue}(p)$ est renommé en $i[\alpha_0]$ et les coups qu'il justifie sont modifiés en conséquence).*

La fonction $\mathrm{vue}^{\perp}$ rétabli simplement la modification des noms.

Les stratégies de ces jeux permettent de représenter des fonctions partielles, comme nous l'avons déjà illustré avec les jeux $\mathcal{J}_{\mathbb{N}}$, $\mathcal{J}_{\mathbb{N}\to\mathbb{N}}$ et $\mathcal{J}_{(\mathbb{N}\to\mathbb{N})\mathbb{N}}$.

Définition 5.26. *Une stratégie s dans le jeu $\mathcal{J}_\tau$ représente une fonction partielle de type τ dans ce sens :*

— *Si $\tau = \mathbb{N}$, alors s représente $n \in \mathbb{N}$ si $s(q) = r_n$ et représente $\perp$ si $s(q)$ n'est pas défini.*

— *Si $\tau = \sigma \to \mathbb{N}$, alors s représente $F : \tau$ si pour toute stratégie s' représentant une fonction $f : \sigma$, F est bien définie en f si et seulement si la confrontation entre s et s' termine et dans ce cas la réponse finale représente $F(f)$.*

— *Si $\tau = \sigma_1 \times \sigma_2$, alors s représente le couple de fonctions représentées par ses restrictions respectives aux jeux $\mathcal{J}_{\sigma_1}$ et $\mathcal{J}_{\sigma_1}$ (voir remarque 5.24).*

Remarque 5.24. *Une stratégie dans le jeu $\mathcal{J}_{\sigma\times\tau}$ peut être décomposée naturellement en une stratégie dans $\mathcal{J}_\sigma$ et une stratégie dans $\mathcal{J}_\tau$ représentant les projections respectives de la fonction calculée. En effet, comme les parties sont justifiées, et les arènes $\mathcal{A}_\sigma$ et $\mathcal{A}_\tau$ ne sont pas connectées dans $\mathcal{A}_{\sigma\times\tau}$, une partie se joue nécessairement exclusivement dans une seule de ces deux arènes. Dans ce cas, la complexité (respectivement, la taille) de chaque stratégie est la composition de la projection correspondante avec la complexité (respectivement, la taille) de la stratégie initiale. Inversement, une stratégie dans $\mathcal{J}_\sigma$ et une stratégie dans le jeu $\mathcal{J}_\tau$ définissent une unique stratégie dans $\mathcal{J}_{\sigma\times\tau}$ représentant la paire correspondante, dont la complexité (respectivement, la taille) est le produit de leurs complexités (respectivement, leurs tailles).*

En termes de calculabilité, ces jeux permettent de capturer exactement PCF (les résultats originels montrent même que ce modèle est pleinement adéquat, mais cela ne nous est pas utile ici).

Théorème 5.2 ([Nic94 ; HO00])**.** *Les fonctions partielles représentées par des stratégies innocentes calculables sont exactement les fonctions définissables dans* PCF*.*

5.3.3 Fonctions calculables en temps polynomial d'ordre supérieur

Les jeux innocents sont bien des jeux séquentiels effectifs, on peut donc appliquer les définitions de la section 5.2 à leurs stratégies. Comme nous l'avons déjà fait remarquer, nous ne pouvons pas définir la complexité de toutes les fonctions de PCF. Déjà, seule une sous-classe de fonctions héréditairement totales de PCF a une taille bien définie. Nous pouvons maintenant appliquer tous les outils définis précédemment aux stratégies innocentes pour définir l'ensemble des fonctions de PCF calculables en temps polynomial.

Définition 5.27. *Soit* POLY_τ *l'ensemble des fonctions de type* τ *de* PCF *représentées par une stratégie calculable en temps polynomial.*

Remarque 5.25. *L'ensemble des fonctions calculables en temps polynomial est une classe de fonctions héréditairement totales (selon la définition 4.1).*

Les fonctions polynomiales ainsi définies correspondent aux fonctions polynomiales usuelles à l'ordre 1.

Proposition 5.14. *Une fonction d'ordre 1 est calculable en temps polynomial si et seulement si elle est représentée par une stratégie de* $\mathcal{J}_{\mathbb{N}\to\mathbb{N}}$ *calculable en temps polynomial (autrement dit,* $\text{FPTIME} = \text{POLY}_{\mathbb{N}\to\mathbb{N}}$*).*

La correspondance est aussi valable à l'ordre 2.

Proposition 5.15. *Une fonction d'ordre 2 est calculable en temps polynomial si et seulement si elle est représentée par une stratégie de* $\mathcal{J}_{(\mathbb{N}\to\mathbb{N})\to\mathbb{N}}$ *calculable en temps polynomial (autrement dit,* $\text{FPTIME}_2 = \text{POLY}_{(\mathbb{N}\to\mathbb{N})\to\mathbb{N}}$*).*

Les deux propositions précédentes sont déjà impliquées par les exemples 5.22 et 5.23 et la proposition 5.12. On peut cependant obtenir des preuves plus simples en remarquant que le fonctionnement des machines sur les jeux $\mathcal{J}_{\mathbb{N}\to\mathbb{N}}$ et $\mathcal{J}_{(\mathbb{N}\to\mathbb{N})\to\mathbb{N}}$ est très proche de celui des machines de Turing et des machines à oracles usuelles.

Proposition 5.16. *Pour tout type* τ*, la complexité de la fonction identité de type* $\tau \to \tau$ *est linéaire.*

Preuve

La fonction identité est représentée par la stratégie *copycat*, qui à chaque coup de l'adversaire dans l'une des deux copies de l'arène $\mathcal{A}_\tau$ le répète dans la seconde arène. Cette stratégie est bien innocente (en particulier, elle ne dépend que du dernier coup joué). En particulier, si on note τ sous forme normale, on obtient $\tau = (\prod_i \sigma_i) \to \mathbb{N}$ (le cas où τ est un produit est similaire). Comme nous l'avons vu dans la remarque 5.24, une stratégie s dans le sous-jeu $\mathcal{J}_{\tau\times(\prod_i \sigma_i)}$ peut être décomposée comme une stratégie s_1 dans $\mathcal{J}_\tau$ et une stratégie s_2 dans $\mathcal{J}_{(\prod_i \sigma_i)}$. La stratégie *copycat* simule donc la confrontation entre ces deux stratégies. Comme elle répète chaque coup joué, la taille de l'historique face à s est donc $2 \cdot H(s_1, s_2)$ qui est borné par $2 \cdot \|s_1\|_{\mathcal{J}_\tau}(\|s_2\|_{\mathcal{J}_{(\prod_i \sigma_i)}})$. Autrement dit, sa complexité est bornée par $\lambda t.2 \cdot (\pi_1 \circ t)(\pi_2 \circ t)$, ce qui est équivalent (en curryfiant) à $\lambda t.2 \cdot t$. □

Corollaire 5.3.

La complexité du constructeur de paires de type $\tau \to \sigma \to (\tau \times \sigma)$ *et des projections associées de types respectifs* $\tau \times \sigma \to \tau$ *et* $\tau \times \sigma \to \sigma$ *est linéaire.*

Preuve

Ces fonctions peuvent en effet être vues comme des cas particulier de la fonction identité. □

Corollaire 5.4. *Pour tous types finis σ et τ, la fonction application de type $(\sigma \to \tau) \times \sigma \to \tau$ est calculable en temps polynomial.*

PREUVE

C'est aussi un cas particulier de fonction identité. Sa complexité est donc, d'après la preuve de la proposition 5.16 bornée par $\lambda(x, y).2 \cdot (x\ y)$. La complexité de la fonction application est donc la fonction application elle-même (à une constante multiplicative près). □

Ce corollaire est une version uniforme du suivant.

Corollaire 5.5. *La classe* POLY *est stable par composition. Plus précisément, si F est calculable en temps P_1 et f en temps P_2, alors $F(f)$ est calculable en temps $2 \cdot P_1(P_2)$.*

Comme on l'attend d'un analogue de FPTIME d'ordre supérieur, cette classe étend BFF à tout ordre.

Théorème 5.3. *Toute fonction de* BFF *a une stratégie innocente polynomiale (i.e.* BFF $\subseteq$ POLY*).*

PREUVE

Il suffit de montrer que tout terme de PV$^\omega$ est calculable en temps polynomial relativement à ses variables libres. Les corollaires 5.4 et 5.3 traitent les cas de l'application et des paires, et la proposition 5.14 celui des fonctions calculables en temps polynomial d'ordre 1. Il reste le cas de l'opérateur $\mathcal{R}$ de récursion d'ordre 2, qui est en fait une fonction polynomiale d'ordre 2, ce qui est couvert par la proposition 5.15 (plus précisément, sa complexité est de l'ordre de $\lambda n, G, Hn_0 \mapsto n \cdot G(H(n_0))$). □

Cependant, à partir de l'ordre 3, POLY contient strictement BFF.

Exemple 5.26. *La fonction Ψ définie dans l'exemple 4.3 n'appartient pas à* BFF*, mais possède une stratégie innocente polynomiale. Celle-ci est d'ailleurs égale à celle représentant Φ, excepté lorsqu'elle fournit une réponse finale (r_{2^x} à la place de r_1). Plus précisément, la complexité de Φ est de l'ordre de $\lambda G, n.G(\lambda y.y + x) + G(\lambda x.1)$ et celle de Ψ est bornée par le double de celle-ci. Ces stratégies commencent par simuler l'évaluation de l'application de leur premier argument à la fonction f_x. La complexité de celle-ci étant de l'ordre de $\lambda y.y + x$, le temps pris par cette étape est de l'ordre de $G(\lambda y.y + x)$, où G est la taille de la fonction F d'entrée. Puis elles font de même en l'évaluant sur la fonction nulle (ce qui prend un temps $G(\lambda x.1)$), puis compare les résultats (ce qui double au plus le temps de calcul). Enfin, si ils sont égaux, elles répondent le coup correspondant à 0, sinon, celui correspondant à 1 ou 2^x respectivement. Dans le second cas, le calcul prend un temps de l'ordre de $|2^x|$, mais comme nous l'avions décrit intuitivement dans l'exemple originel, si les deux évaluations fournissent un résultat différent, c'est nécessairement que la stratégie jouant F a joué le coup r'_{2^x}, auquel cas $|2^x| \leq |r'_{2^x}| \leq G(\lambda y.y + x)$ et la stratégie pour Ψ est donc bien polynomiale.*

5.4 Applications et développements possibles

Dans ce chapitre nous avons décrit une certaine classe de jeux (les jeux séquentiels) munis de conditions nécessaires (*i.e.* les conditions « effectives ») pour parler de calculabilité et de complexité. Nous avons défini une notion de taille pertinente pour les stratégies dans ces jeux, qui est le point important qui nous a ensuite permis de définir une notion de complexité intéressante par deux approches différentes, et en particulier une notion de stratégie calculable en temps polynomial à l'aide de polynômes d'ordre supérieur. Ces résultats s'appliquent en particulier aux jeux innocents qui ont permis de caractériser les fonctions de PCF et nous avons donc défini une classe de fonctions calculables en temps polynomial d'ordre supérieur vérifiant les conditions minimales que l'on peut attendre d'une telle classe (notamment énumérées par Bellantoni [Bel90]). Celle-ci contient bien BFF à tout ordre et coïncide avec les classes usuelles aux ordres 1 et 2. De plus, elle semble ne pas souffrir de certains inconvénients de BFF.

Il serait maintenant intéressant de fournir des arguments supplémentaires visant à justifier davantage la pertinence de la classe POLY. En particulier, il serait intéressant de voir si l'on peut remplacer l'opérateur de récursion d'ordre 2 de PV^ω par un opérateur plus puissant permettant de caractériser précisément POLY (et non plus BFF). On peut aussi essayer de rattacher cette définition à d'autres mesures de temps de calcul, par exemple la complexité dérivationnelle (mesurant le nombre de réductions à partir d'un terme dans un langage fonctionnel) [San90 ; DPR13 ; DR07 ; DM10].

Il ne faut cependant pas oublier que celle-ci ne concerne que les jeux innocents, et qu'une autre classe de jeux permettrait de définir une autre classe de fonctions ou d'objets calculables. En effet, la sémantique des jeux ne permet pas uniquement de représenter des objets purement fonctionnels. Par exemple, un jeu similaire à $\mathcal{J}_{\mathbb{N}\to\mathbb{N}}$ permet de représenter des nombres réels par des jeux de manière analogue à la représentation de Cauchy : s représente $x \in [0,1]$ si $s(q') = q$ et pour tout $n \in \mathbb{N}, s(q', q, r_n) = r_{q_n}$ où $q_n \in \mathbb{Q}$ vérifie $|x - q_n| \leq 2^{-n}$. Un tel jeu (muni d'un encodage adéquat) permet alors de définir les mêmes notions de calculabilité et de complexité que celles définies dans le chapitre 2. De plus, de tels jeux pourraient permettre de définir une notion de complexité significative pour des espaces tels que ceux décrits dans le théorème 3.10, pour lesquels la complexité d'ordre 2 n'est pas satisfaisante.

D'autres applications sont possibles, notamment grâce au modèle de machines à jeux qui semble suffisamment général pour permettre de définir naturellement une notion de complexité en espace ou non-déterministe. Nous pouvons d'ailleurs faire ici un parallèle avec les résultats du chapitre 3. En effet, les théorèmes 3.6 et 3.8 peuvent être interprétés ainsi : une fonction est calculable en temps polynomial (respectivement, polynomial non-déterministe) relativement à un oracle si et seulement si sa taille (respectivement, « taille non-déterministe ») est polynomiale (ces notions de taille étant définies dans ce cas à partir des ensembles importants et suffisants). Dans

le cas déterministe, nous avons lié la complexité déterministe à la taille d'une stratégie (dans le corollaire 5.2), et l'historique d'une confrontation décrit intuitivement un ensemble de « points important ». Il semble possible de définir de même une notion de taille non-déterministe (à partir de stratégies non-déterministes), comme nous l'avons déjà remarqué dans le chapitre précédent à propos des voisinages de Kreisel (remarque 4.6).

Enfin, cette notion de complexité ne concerne que les jeux séquentiels. On peut alors se demander s'il est possible d'adapter ces définitions à des classes de jeux plus générales, notamment des jeux non alternants (pour étudier des processus parallèles), ou sur des arènes cycliques (qui permettent entre autres de définir des structures inductives ou co-inductives).

Index des notations

H, 92
Lip_1, 32
$\|\|\|_1$, 32
$\mathcal{V}$, 70
$\mathcal{A}$, 81
$\vdash$, 81
$\mathfrak{C}$, 81
δ_C, 21
$\delta_{[0,1]}$, 21
$\delta_{\pm}$, 21
$\delta_{\Box}$, 28
δ_b, 21
$\mathrm{d}_{,F}\alpha$, 35
dom, 20
enc, 89
$\equiv_c$, 25
$\equiv_p$, 26
$\equiv_t$, 22
$\eta_{,,}$, 51
$\mathfrak{F}$, 81
$\mathfrak{I}$, 81
$\mathcal{J}$, 88
$\langle\rangle$, 20
$\leq_c$, 25
$\leq_p$, 26
$\leq_t$, 22
D_F, 37
$\mathfrak{P}$, 84
$\hookrightarrow$, 20
π_i, 94
por, 75
$\mathfrak{Q}$, 81
$\mathcal{I}$, 40
$\mathcal{R}$, 71
$\mathfrak{R}$, 81
$\preccurlyeq$, 94
$\mathcal{A}^i$, 88
Supp , 32
$||$, 8, 94
$||_1$, 13
$\|\|\|_{\mathcal{J}}$, 95
$\tau_{\mathcal{J}}$, 94
$\|\|\|_\infty$, 32
vue, 89
$\mathrm{vue}^\perp$, 89
$h_{\alpha,r}$, 35
$s[]$, 92
C, 68
RC, 68
$\textsc{pv}^\omega$, 71
BFF, 71
$\textsc{bff}_2$, 17
$\textsc{fptime}_2$, 15
POLY, 107
PCF, 75

Bibliographie

[AJM13] Samson Abramsky, Radha Jagadeesan et Pasquale Malacaria. "Full Abstraction for PCF". Dans : *CoRR* abs/1311.6125 (2013) (cité page 76).

[BC82] Gérard Berry et Pierre-Louis Curien. "Sequential algorithms on concrete data structures". Dans : *Theoretical Computer Science* 20.3 (1982), pages 265–321 (cité page 75).

[BC92] Stephen Bellantoni et Stephen A. Cook. "A new recursion-theoretic characterization of the polytime functions". Dans : *Computational complexity* 2.2 (1992), pages 97–110 (cité pages 1, 73).

[Bel90] Stephen Bellantoni. "Comments On Two Notions of Higher Type Computability." Dans : *Unpublished notes* (1990) (cité pages 71, 109).

[BGP11] Olivier Bournez, Daniel S. Graça et Amaury Pouly. "Solving Analytic Differential Equations in Polynomial Time over Unbounded Domains". Dans : *MFCS*. Édité par Filip Murlak et Piotr Sankowski. Tome 6907. Lecture Notes in Computer Science. Springer, 2011, pages 170–181 (cité page 19).

[BGP12] Olivier Bournez, Daniel S. Graça et Amaury Pouly. "On the complexity of solving initial value problems". Dans : *ISSAC*. Édité par Joris van der Hoeven et Mark van Hoeij. International Symposium on Symbolic and Algebraic Computation, ISSAC'12. Grenoble, France : ACM, 2012, pages 115–121 (cité pages 1, 19).

[BGP13] Olivier Bournez, Daniel S. Graça et Amaury Pouly. "Turing Machines Can Be Efficiently Simulated by the General Purpose Analog Computer". Dans : *TAMC*. Édité par T. -H. Hubert Chan, Lap Chi Lau et Luca Trevisan. Tome 7876. Lecture Notes in Computer Science. Springer, 2013, pages 169–180 (cité page 19).

[BH05] Olivier Bournez et Emmanuel Hainry. "Elementarily computable functions over the real numbers and -sub-recursive functions". Dans : *Theoretical Computer Science* 348.2-3 (2005). Automata, Languages and Programming : Algorithms and Complexity (ICALP-A 2004), pages 130–147 (cité page 1).

[BM10] Patrick Baillot et Damiano Mazza. "Linear logic by levels and bounded time complexity ". Dans : *Theoretical Computer Science* 411.2 (2010), pages 470–503 (cité page 1).

[BMM05] Guillaume Bonfante, Jean-Yves Marion et Jean-Yves Moyen. *Quasi-interpretation : a way to control ressources*. Rapport technique. 2005, page 35 (cité page 1).

[Bou+07] Olivier Bournez, Manuel L. Campagnolo, Daniel S. Graça et Emmanuel Hainry. "Polynomial differential equations compute all real computable functions on computable compact intervals". Dans : *Journal of Complexity* 23.3 (2007), pages 317–335 (cité pages 1, 19).

[BSS89] Lenore Blum, Mike Shub et Steve Smale. "On a theory of computation and complexity over the real numbers : NP-completeness, recursive functions and universal machines". Dans : *bullamer* 21.1 (1989), pages 1–46 (cité page 19).

[CK89] Stephen A. Cook et Bruce M. Kapron. "Characterizations of the basic feasible functionals of finite type". Dans : *Foundations of Computer Science, IEEE Annual Symposium on* (1989), pages 154–159 (cité page 72).

[Cob65] Alan Cobham. "The Intrinsic Computational Difficulty of Functions". Dans : (1965), page 24 (cité pages 1, 16, 71).

[CU93] Stephen Cook et Alasdair Urquhart. "Functional interpretations of feasibly constructive arithmetic". Dans : *Annals of Pure and Applied Logic* 63.2 (1993), pages 103–200 (cité pages 2, 17, 71).

[Cur93] Pierre-Louis Curien. "Sequential Algorithms". Dans : *Categorical Combinators, Sequential Algorithms, and Functional Programming*. Birkhäuser Boston, 1993. Chapitre Progress in Theoretical Computer Science, pages 159–250 (cité page 75).

[DM10] Ugo Dal Lago et Simone Martini. "Derivational Complexity Is an Invariant Cost Model". Dans : *Foundational and Practical Aspects of Resource Analysis*. Édité par Marko van Eekelen et Olha Shkaravska. Tome 6324. Springer Berlin Heidelberg, 2010. Chapitre Lecture Notes in Computer Science, pages 100–113 (cité page 109).

[DPR13] Norman Danner, Jennifer Paykin et James S. Royer. "A static cost analysis for a higher-order language". Dans : *PLPV*. Édité par Matthew Might, David Van Horn, Andreas Abel et Tim Sheard. Proceedings of the 7th Workshop on Programming languages meets program verification, PLPV 2013, Rome, Italy, January 22, 2013. ACM, 2013, pages 25–34 (cité page 109).

[DR07] Norman Danner et James S. Royer. "Adventures in time and space". Dans : *Logical Methods in Computer Science* 3.1 (2007) (cité page 109).

[Fér+10] Hugo Férée, Emmanuel Hainry, Mathieu Hoyrup et Romain Péchoux. "Interpretation of Stream Programs : Characterizing Type 2 Polynomial Time Complexity". Dans : *ISAAC 2010*. Édité par Otfried Cheong, Kyung-Yong Chwa et Kunsoo Park. Tome 6506. Lecture Notes in Computer Science. Springer, 2010. Chapitre Lecture Notes in Computer Science, pages 291–303 (cité page 4).

[Fér+14] Hugo Férée, Emmanuel Hainry, Mathieu Hoyrup et Romain Péchoux. "Characterizing polynomial time complexity of stream programs using interpretations". Dans : *Theoretical Computer Science* (2014) (cité pages 1, 4).

[FGH14] Hugo Férée, Walid Gomaa et Mathieu Hoyrup. "Analytical properties of resource-bounded real functionals ". Dans : *Journal of Complexity* 30.5 (Jul 2014), pages 647–671 (cité pages 4, 31).

[FH13] Hugo Férée et Mathieu Hoyrup. "Higher-order complexity in analysis". Dans : *CCA 2013 : Computability and Complexity in Analysis*. Nancy, France : FernUniversität in Hagen, 2013 (cité pages 4, 59).

[FHG13] Hugo Férée, Mathieu Hoyrup et Walid Gomaa. "On the Query Complexity of Real Functionals". Dans : *LICS - 28th ACM/IEEE Symposium on Logic in Computer Science*. New Orleans, États-Unis, Jan 2013, pages 103–112 (cité pages 4, 31).

[Fri84] Harvey Friedman. "The Computational Complexity of Maximization and Integration". Dans : *Advances in Math.* 53.1 (1984), pages 80–98 (cité page 27).

[GC11] Murdoch J. Gabbay et Vincenzo Ciancia. "Freshness and name-restriction in sets of traces with names". Dans : *Foundations of software science and computation structures, 14th International Conference (FOSSACS 2011)*. Tome 6604. Lecture Notes in Computer Science. Springer, 2011, pages 365–380 (cité page 86).

[GG12] Murdoch J. Gabbay et Dan R. Ghica. "Game Semantics in the Nominal Model". Dans : (2012), pages 173–189 (cité pages 79, 84).

[GH77] Robin O. Gandy et John Martin Elliott Hyland. "Computable and recursively countable functions of higher type". Dans : *Studies in Logic and the Foundations of Mathematics* 87 (1977), pages 407–438 (cité page 67).

[Ghi05] Dan R. Ghica. "Slot games : a quantitative model of computation". Dans : *POPL*. Édité par Jens Palsberg et Martín Abadi. Proceedings of the 32nd ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages. ACM, 2005, pages 85–97 (cité page 79).

[Ghi09] Dan R. Ghica. "Applications of Game Semantics : From Program Analysis to Hardware Synthesis". Dans : *LICS*. Proceedings of the 24th Annual IEEE Symposium on Logic in Computer Science, LICS 2009, 11-14 August 2009, Los Angeles, CA, USA. IEEE Computer Society, 2009, pages 17–26 (cité page 76).

[Gir98] Jean-Yves Girard. "Light linear logic". Dans : (1998), pages 145–176 (cité page 1).

[HHM07] Russell Harmer, Martin Hyland et Paul-André Melliès. "Categorical Combinatorics for Innocent Strategies". Dans : *LICS*. 22nd IEEE Symposium on Logic in Computer Science. IEEE Computer Society, 2007, pages 379–388 (cité page 79).

[HO00] John. Martin. Elliott. Hyland et C. -H. Luke Ong. "On Full Abstraction for PCF : I, II, and III". Dans : *Inf. Comput.* 163.2 (2000), pages 285–408 (cité pages 76, 105, 106).

[Hyl78] John Martin Elliott Hyland. "The intrinsic recursion theory on the countable or continuous functionals ". Dans : *Generalized Recursion Theory II Proceedings of the 1977 Oslo Symposium*. Édité par Robin O. Gandy J. E. Fenstad et G. E. Sacks. Tome 94. Elsevier, 1978. Chapitre Studies in Logic and the Foundations of Mathematics, pages 135–145 (cité page 69).

[IRK01] Robert J. Irwin, James S. Royer et Bruce M. Kapron. "On characterizations of the basic feasible functionals (Part I)". Dans : *J. Funct. Program.* 11.1 (2001), pages 117–153 (cité page 73).

[IRK02] Robert J. Irwin, James S. Royer et Bruce M. Kapron. "On Characterizations of the Basic Feasible Functionals (Part II)". Unpublished. 2002 (cité pages 72, 73, 79, 103).

[IS04] Aleksandar Ignjatovic et Arun Sharma. "Some Applications of Logic to Feasibility in Higher Types". Dans : *ACM Trans. Comput. Logic* 5.2 (04 2004), pages 332–350 (cité page 13).

[Kaw+12a] Akitoshi Kawamura, Norbert Th. Müller, Carsten Rösnick et Martin Ziegler. "Parameterized Uniform Complexity in Numerics : from Smooth to Analytic, from NP-hard to Polytime". Dans : *Computing Research Repository* abs/1211.4974 (2012) (cité page 59).

[Kaw+12b] Akitoshi Kawamura, Hiroyuki Ota, Carsten Rösnick et Martin Ziegler. "Computational Complexity of Smooth Differential Equations". Dans : *MFCS*. Édité par Branislav Rovan, Vladimiro Sassone et Peter Widmayer. Tome 7464. Lecture Notes in Computer Science. Springer, 2012, pages 578–589 (cité page 20).

[KC10] Akitoshi Kawamura et Stephen Cook. "Complexity theory for operators in analysis". Dans : *Proceedings of the 42nd ACM Symposium on Theory of Computing*. STOC '10. Cambridge, Massachusetts, USA : ACM, 2010, pages 495–502 (cité pages 20, 26, 28, 49, 59).

[KC96] Bruce M. Kapron et Stephen A. Cook. "A New Characterization of Type-2 Feasibility". Dans : *SIAM Journal on Computing* 25.1 (1996), pages 117–132 (cité pages 1, 13, 14, 16, 17, 103).

[Kec95] Alexander S. Kechris. *Classical Descriptive Set Theory*. Springer, 1995 (cité page 60).

[KF82] Ker-I Ko et Harvey Friedman. "Computational Complexity of Real Functions". Dans : *Theoretical Computer Science* 20.3 (1982), pages 323–352 (cité page 32).

[Kle59a] Stephen Cole Kleene. "Countable functionals". Dans : *Constructivity in Mathematics* (1959), pages 81–100 (cité page 67).

[Kle59b] Stephen Cole Kleene. "Recursive functionals and quantifiers of finite types I". Dans : *Transactions of the American Mathematical Society* (1959), pages 1–52 (cité pages 66, 67).

[Kle63] Stephen Cole Kleene. "Recursive functionals and quantifiers of finite types II". Dans : *Transactions of the American Mathematical Society* (1963), pages 106–142 (cité page 66).

[Ko82] Ker-I. Ko. "The Maximum Value Problem and NP Real Numbers". Dans : *J. Comput. Syst. Sci.* 24.1 (1982), pages 15–35 (cité pages 27, 29, 49).

[Ko91] Ker-I. Ko. *Complexity Theory of Real Functions*. Birkhauser Boston Inc. Cambridge, MA, USA, 1991 (cité pages 1, 11, 20, 29, 58).

[Kre59] Georg Kreisel. "Interpretation of analysis by means of constructive functionals of finite types". Dans : *Constructivity in Mathematics*. Édité par A. Heyting. Studies in Logic and the Foundations of Mathematics. Proc. Colloq., Amsterdam, Aug. 26–31, 1957. Amsterdam : North-Holland, 1959, pages 101–128 (cité page 67).

[Laf04] Yves Lafont. "Soft linear logic and polynomial time ". Dans : *Theoretical Computer Science* 318.1–2 (2004). Implicit Computational Complexity, pages 163–180 (cité page 1).

[Lei95] Daniel Leivant. "Ramified Recurrence and Computational Complexity I : Word Recurrence and Poly-time". Dans : *Feasible Mathematics II*. Édité par Peter Clote et JeffreyB. Remmel. Tome 13. Birkhäuser Boston, 1995, pages 320–343 (cité page 73).

[LL08] Ugo Dal Lago et Olivier Laurent. "Quantitative Game Semantics for Linear Logic". Dans : *CSL*. Édité par Michael Kaminski et Simone Martini. Tome 5213. Lecture Notes in Computer Science. Springer, 2008, pages 230–245 (cité page 79).

[LM93] Daniel Leivant et Jean-Yves Marion. "Lambda calculus characterizations of poly-time". Dans : *Typed Lambda Calculi and Applications*. Édité par Marc Bezem et JanFriso Groote. Tome 664. Springer, 1993, pages 274–288 (cité pages 1, 73).

[Lon00] John R. Longley. "Notions of computability at higher types I". Dans : *Logic Colloquium*. Tome 19. 2000, pages 32–142 (cité pages 2, 65, 69).

[Meh76] Kurt Mehlhorn. "Polynomial and abstract subrecursive classes". Dans : *J. Comput. Syst. Sci.* 12.2 (apr 1976), pages 147–178 (cité pages 13, 16).

[Mur05] Andrzej S. Murawski. "Games for complexity of second-order call-by-name programs ". Dans : *Theoretical Computer Science* 343.1–2 (2005). Game Theory Meets Theoretical Computer Science Game Theory Meets Theoretical Computer Science, pages 207–236 (cité page 79).

[Nic+96] Hanno Nickau, Doktors Der Naturwissenschaften, Dipl. –math Hanno Nickau, Angenommen Vom Fachbereich et al. *Hereditarily Sequential Functionals : A Game-Theoretic Approach to Sequentiality*. 1996 (cité pages 76, 105).

[Nic94] Hanno Nickau. "Hereditarily sequential functionals". Dans : *Logical Foundations of Computer Science*. Édité par Anil Nerode et Yu. V. Matiyasevich. Tome 813. Springer Berlin Heidelberg, 1994. Chapitre Lecture Notes in Computer Science, pages 253–264 (cité pages 76, 105, 106).

[Nor80] Dag Normann. "The countable functionals". Dans : *Recursion on the Countable Functionals*. Tome 811. Springer Berlin Heidelberg, 1980. Chapitre Lecture Notes in Mathematics, pages 23–48 (cité page 69).

[Plo77] G. D. Plotkin. "LCF considered as a programming language ". Dans : *Theoretical Computer Science* 5.3 (1977), pages 223–255 (cité page 75).

[Rud91] Walter Rudin. *Functional analysis*. Second. International Series in Pure and Applied Mathematics. New York : McGraw-Hill Inc., 1991, pages xviii+424 (cité page 35).

[San90] David Sands. "Complexity Analysis for a Lazy Higher-Order Language". Dans : *ESOP*. Édité par Neil D. Jones. Tome 432. Lecture Notes in Computer Science. Springer, 1990, pages 361–376 (cité page 109).

[Saz76] V. Yu. Sazonov. "Degrees of parallelism in computations". Dans : *Mathematical Foundations of Computer Science 1976*. Édité par Antoni Mazurkiewicz. Tome 45. Springer Berlin Heidelberg, 1976, pages 517–523 (cité page 75).

[Sch02] Matthias Schröder. “Extended admissibility”. Dans : *Theoretical Computer Science* 284.2 (2002), pages 519–538 (cité page 59).

[Sco93] Dana S. Scott. “A type-theoretical alternative to ISWIM, CUCH, OWHY ”. Dans : *Theoretical Computer Science* 121.1–2 (1993), pages 411–440 (cité page 74).

[Set93] Anil Seth. “Some Desirable Conditions for Feasible Functionals of Type 2”. Dans : *LICS - 8th ACM/IEEE Symposium on Logic in Computer Science*. Jun 1993, pages 320–331 (cité pages 17, 71).

[Set94] Anil Seth. “Complexity theory of higher type functionals”. Dans : *University of Bombay* (1994) (cité page 73).

[Set95] Anil Seth. “Turing Machine Characterizations of Feasible Functionals of All Finite Types”. Dans : *Feasible Mathematics II* (1995), pages 407–428 (cité page 73).

[Sha41] Claude E. Shannon. “Mathematical theory of the differential analyzer”. Dans : *J. Math. Phys. MIT* 20 (1941), pages 337–354 (cité page 19).

[Tur36] Alan Turing. “On computable numbers, with an application to the Entscheidungsproblem”. Dans : *Proceedings of the London Mathematical Society* 2, 42 (1936), pages 230–265 (cité page 19).

[Wei00] Klaus Weihrauch. *Computable Analysis*. Berlin : Springer, 2000 (cité pages 1, 8, 11, 19, 20, 23, 24, 59).

Résumé

Alors que la complexité des fonctions d'ordre 1 est bien définie et étudiée, ils n'existe pas de notion satisfaisante à tout ordre. Une telle théorie existe déjà à l'ordre 2 et permet de définir une classe analogue aux fonctions calculables en temps polynomial usuelles. Cela est tout particulièrement intéressant dans le domaine de l'analyse récursive où l'on peut représenter, entre autres, les nombres et les fonctions réelles par des fonctions d'ordre 1. On peut alors remarquer un lien fort entre calculabilité et continuité, et aussi rapprocher la complexité avec certaines propriétés analytiques, ce que nous illustrons dans le cas des opérateurs réels. Nous prouvons cependant que, du point de vue de la complexité, les fonctions d'ordre 1 ne permettent pas de représenter fidèlement certains espaces mathématiques. Ce résultat appuie tout particulièrement la nécessité d'une théorie de la complexité d'ordre supérieur.

Nous développons alors un modèle de calcul basé sur la sémantique des jeux, où l'application de deux fonctions est représentée par la confrontation de deux stratégies dans un jeu. En définissant la taille de telles stratégies, nous pouvons déduire une notion robuste et pertinente de complexité pour ces stratégies et donc pour les fonctions d'ordre supérieur. Nous définissons aussi une classe de fonctions calculables en temps polynomial qui paraît être un bon candidat pour définir une classe de complexité raisonnable à tout ordre.

Mots-clés: complexité, analyse récursive, ordre supérieur.

Abstract

While first order complexity is well defined and studied, higher order lacks a satisfactory notion of complexity. Such a theory already exists at order 2 and provides a complexity class analogue to usual polynomial time computable functions. This is already especially interesting in the domain of computable analysis, where real numbers or real functions for example can be represented by first order functions. In particular, there is a clear link between computability and continuity, and we also illustrate in the case of real operators that complexity can be related to some analytical properties. However, we prove that, from a complexity point of view, some mathematical spaces can not be faithfully represented by order 1 functions and require higher order ones. This result underlines that there is a need to develop a notion of complexity at higher types which will be, in particular but not only, useful to computable analysis.

We have developed a computational model for higher type sequential computations based on a game semantics approach, where the application of two functions is represented by the game opposing two strategies. By defining the size of such strategies, we are able to define a robust and meaningful notion of complexity at all types, together with a class of polynomial time computable higher order functionals which seems to be a good candidate for a class of feasible functionals at higher types.

Keywords: complexity, computable analysis, higher order.