FREC, délivrable 6 :

Cost automata over words : algorithms and logical formalisms

(Tâche 4)

Ce document correspond à la thèse d'habilitation à diriger des recherches de Thomas Colcombet. Il contient une description poussée des derniers avancements concernant les fondements des automates à coût sur les mots.

Université Paris Diderot – Paris 7
Laboratoire d'Informatique Algorithmique : Fondements et Applications

---

Fonctions régulières de coût

Thèse présentée pour l'obtention du diplôme d'

HABILITATION À DIRIGER LES RECHERCHES

à l'École Doctorale de Sciences Mathématiques de Paris Centre

Par

Thomas Colcombet

Soutenue publiquement
le 25 mars 2013
devant le jury constitué de :

Prof. Damian Niwiński	
Prof. Dominique Perrin	
D.R. Jean-Éric Pin	Rapporteur
D.R. Philippe Schnoebelen	
Prof. Howard Straubing	
Prof. Wolfgang Thomas	Rapporteur
D.R. Igor Walukiewicz	Rapporteur

Table des matières

Première partie

Histoire et présentation

Chapitre 1

Introduction

L'objet de cette thèse est l'étude des fonctions régulières de coût. Il s'agit d'une extension quantitative de la notion, classique, de langages réguliers, qui en préserve certaines propriétés clefs. Plus précisément les fonctions régulières de coût ont été introduites comme une théorie unifiant la théorie classique des langages réguliers avec les travaux, moins connus, sur les automates de distance et l'algèbre des matrices sur le semi-anneau tropical.

Au cours de cette introduction, nous présentons à la section 1.1 quelques résultats importants de la théorie des langages. Les automates de distance et la branche de recherche dont ils sont la source sont le sujet de la section 1.2. La section 1.3 présente une certaine extension de la logique monadique sur les mots infinis, $\mathrm{MSO} + \mathbb{U}$, dont l'étude a amené au développement de la théorie des fonctions régulières de coût. Nous présenterons enfin les fonctions régulières de coût proprement dites au cours de la section 1.4. L'avant-dernière section, section 1.5 présente brièvement l'approche par mots profinis due à Szymon Toruńczyk. Enfin, la section 1.6 énonce quelques problèmes plus ou moins classiques qui peuvent être abordés en utilisant cette théorie.

1.1 La théorie classique des langages réguliers

Ce premier paragraphe décrit brièvement certaines facettes de la théorie des langages réguliers. Ces résultats nous serviront de points de repère pour le développement des fonctions régulières de coût, qui en étendent certains des plus essentiels.

Dans son acceptation la plus restreinte, un langage est un ensemble de mots finis, c'est à dire un ensemble de séquences finies de symboles appartenant à un alphabet fini. Les questions essentielles que l'on considère sont les suivantes :

- Comment peut-on décrire un langage de manière finie (le langage étant lui même infini) ? En d'autres termes, quels sont les différents modèles d'**accepteurs** de langages ?
- Est-ce que différentes classes d'accepteurs ont le même pouvoir d'expression ?

- Quelles sont les opérations qui peuvent être réalisées sur une classe d'accepteurs donnée ? Par exemple, peut-on effectuer l'union, l'intersection ou le complément de langages simplement en transformant les accepteurs ?
- Que peut-on décider pour les langages acceptés par une classe d'accepteurs ? Typiquement, est-ce qu'un langage est vide, plein (c'est à dire que son complémentaire est vide [1]), infini ? Quelle est la complexité de ces problèmes de décision ?
- Si deux modèles d'accepteurs sont d'expressivités différentes, est-il possible de caractériser l'une de ces classes à l'intérieur de l'autre ?

Les deux premières questions s'apparentent fortement à la théorie de la complexité. Et c'est en effet le cas. L'une des manières d'appréhender a théorie des automates est comme une théorie de la complexité mettant en jeu des machines «particulièrement simples». En raison de cette simplicité, certaines questions éminemment ardues deviennent abordables, comme, le plus souvent, les questions de séparation de classes. En contrepartie, de nouvelles questions qui n'ont pas de sens en théorie de la complexité deviennent le sujet central d'étude. Il s'agit en particulier des problèmes de décision. À titre d'exemple, le test du vide d'un langage, qui est indécidable pour presque toutes les classes de complexité, devient décidable pour la plupart des familles d'automates.

Bien entendu, la question première concerne les modèles de calculs :

Quels sont les accepteurs permettant de décrire un langage ?

Il est possible de classer les différents types d'accepteurs en quatre grandes familles (bien que les distinctions entre ces familles s'estompent parfois, et peuvent être discutées).

Description d'un langage au moyen d'une logique. Il s'agit de se donner une formule dans une logique donnée. Le langage défini par cette formule est alors l'ensemble de structures (des mots par exemple) pour lesquelles cette formule est vraie. Par exemple la formule du premier-ordre

$$\forall x\, a(x) \to \exists y\, x < y \wedge b(y)$$

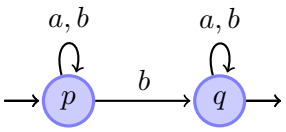
s'interprète en considérant que les variables x, y représentent des positions dans un mot, que $x < y$ signifie que «x est à gauche de y», et $a(x)$ énonce que la lettre à la position x est un a. Ainsi, cette formule représente l'ensemble des mots (sur l'alphabet $\{a, b\}$) tels que toute occurrence d'une lettre a est suivie ultérieurement d'une lettre b. Bien entendu, suivant l'expressivité de la logique que l'on se fixe, les familles de langages obtenues peuvent être différentes. L'exemple ci-dessus fait usage de la logique du premier ordre. La logique du premier ordre autorise de quantifier sur les éléments (*c.-à-d.* les positions dans un mot, les nœuds dans un arbre, les sommets dans un graphe...), d'utiliser tous les connecteurs booléens et de faire référence aux prédicats de la structure (l'ordre sur les positions dans un mot, et la présence de lettres).

Une logique joue un rôle particulier dans cette approche, il s'agit de la logique du second-ordre monadique (appelée simplement **logique monadique** dans la suite). Cette

1. Aussi appelé universel.

logique étend la logique du premier ordre par la possibilité d'effectuer des quantifications sur les ensembles d'éléments (les variables sont alors appelées **monadiques** et sont notées en lettres capitales). Cette logique est plus expressive que la logique du premier ordre. En particulier, elle permet d'exprimer des propriétés «non locales», comme la connectivité dans un graphe ou la parité de la longueur d'un mot, choses toutes deux inexprimables au premier ordre.

Description d'un langage au moyen d'un automate. Un automate est une machine à état fini travaillant sur une structure donnée en entrée. Un automate «construit» une exécution qui doit respecter un certain nombre de contraintes locales énoncées dans une table de transition. L'automate possède également un critère permettant d'identifier certaines exécutions comme acceptantes (ce qui peut également être vu comme une forme de contrainte).



Dans l'automate ci-dessus, une exécution sur un mot u est un chemin partant de l'état p (flèche entrante) et aboutissant dans l'état q (flèche sortante), tel que la suites des étiquettes le long de ce chemin produise le mot u. Une structure est acceptée s'il existe une exécution acceptante de l'automate sur cette structure. Un langage peut alors être décrit par la donnée d'un automate : le langage formé des objets acceptés par l'automate. Dans le cas ci-dessus il s'agit de l'ensemble des mots contenant une occurrence de la lettre b. Les automates existent en de nombreuses variantes : déterministes, non-déterministes, non-ambigus, alternants, boustrophédons (pouvant faire demi-tour), etc...

Description d'un langage au moyen d'une expression rationnelle. Il s'agit de se fixer un certain nombre de langages (le plus souvent les lettres isolées), et un certain nombre d'opérations sur les langages (union, concaténation, étoile de Kleene, et parfois intersection ou complément). Un langage est alors dans la classe s'il peut être obtenu en partant des langages fixés par application de ces opérations. Ainsi, l'ensemble des mots sur $\{a, b\}$ contenant une occurrence de la lettre b est décrit par l'expression :

$$(a + b)^* b a^* \ .$$

Reconnaissabilité de langages au moyen d'un objet algébrique. Les objets sur lesquels nous travaillons sont généralement munis d'une structure algébrique : ainsi les mots sur un alphabet fini sont équipés du produit de concaténation, et ils forment en ce sens un monoïde (*c.-à-d.*, le produit de concaténation est associatif et le mot vide en est un élément neutre). Une partie d'une telle structure algébrique est dite reconnaissable si elle peut être obtenue par image inverse d'un morphisme vers une structure finie. Ainsi, considérons le monoïde à

deux éléments $\{0,1\}$ équipé du produit défini par $11 = 1$ et $00 = 10 = 01 = 0$. Considérons de plus le morphisme ρ des mots sur l'alphabet $\{a,b\}$ défini par $\rho(a) = 1$ et $\rho(b) = 0$. Alors l'ensemble des mots contenant une occurrence de b est simplement $\rho^{-1}(\{0\})$. Ainsi le langage des mots contenant une occurrence de b est reconnaissable. Une définition alternative de la reconnaissabilité fait usage de congruences : un langage est reconnaissable s'il existe une congruence finie pour la loi de concaténation telle que toute intersection d'une classe d'équivalence avec le langage soit triviale (*c.-à-d.* vide ou égale à la classe elle-même). Ainsi, la relation d'équivalence définie par $u \sim v$ si «u contient une occurrence de b si et seulement si v contient une occurrence de b» est une congruence d'index 2. Le langage des mots contenant une occurrence de b est donc reconnaissable.

Ces quatre approches sont de nature différente, et pourtant elles sont reliées de manière forte, ce qu'il est possible de qualifier de **théorème fondamental des langages réguliers** (Kleene, Schützenberger, Myhill, Rabin, Scott, Büchi, Elgot, Trakhtenbrot) :

Théorème 1.1 ([55, 91, 79, 89, 15, 38, 16])**.** *Pour un langage de mots finis, les propriétés suivantes sont équivalentes :*

- *être accepté par une expression régulière,*
- *être accepté par un automate fini déterministe,*
- *être accepté par un automate fini non-déterministe,*
- *être définissable en logique monadique,*
- *être reconnaissable par monoïde fini.*

De plus, ces équivalences sont effectives, et le vide de ces langages est décidable.

Une conséquence clef de ce résultat concerne la décidabilité de la logique monadique. La **théorie** d'une structure, c'est l'ensemble des formules qui sont vraies sur cette structure. La **théorie** d'un ensemble de structures est l'union des théories de ces structures. Ainsi, décider la théorie d'une famille de structures, cela signifie que l'on peut décider, étant donné une formule, s'il existe une structure sur laquelle elle est satisfaite. Nous parlons aussi du problème de **satisfaction**.

Corollaire 1.2. *La théorie monadique des mots finis est décidable.*

Il s'agit d'une conséquence du théorème fondamental, puisque cela correspond à tester le vide du langage défini par la formule. Ce corollaire est un guide essentiel dans la suite de ce travail. Nous ne nous intéressons qu'à des logiques pour lesquelles la théorie est décidable sur une famille de structures données.

Ces résultats furent à l'origine de la très riche théorie des automates (pour une description de l'historique de ces travaux, voir [85]). Citons les principales extensions du théorème fondamental des langages réguliers.

Les langages de mots infinis. Ici, les possibilités sont nombreuses dépendant de la «longueur» de ces mots. Les mots infinis les plus courants ont pour longueur ω. Des longueurs supérieures ont été étudiées, ordinales, dispersées, quelconques dénombrables,

etc... Le fameux théorème de Büchi [16] a initié cette branche en considérant les langages de mots indexés par ω. Les modèles algébriques n'ont été compris que plus tardivement [106, 86, 21]. La limite de la décidabilité dans cette direction se situe à la frontière entre mots de longueur dénombrable et de longueur non-dénombrable (la théorie monadique des mots indexés par l'ordre des réels est en particulier indécidable [95]).

Les langages d'arbres finis. Ces travaux furent initiés par Doner, Thatcher et Wright sur les arbres finis [100, 36].

Les langages d'arbres infinis. Ces arbres apparaissent très naturellement en vérification puisqu'ils permettent de modéliser naturellement le temps discret branchant. Le théorème fondamental correspondant est dû à Rabin [87]. Ce résultat, qui stipule que la théorie monadique des arbres infinis est décidable, est souvent considéré comme l'un des (si n'est le) plus importants résultats de décidabilité en théorie des langages et en vérification. Gurevich et Harrington ont par la suite montré qu'un objet essentiel à la compréhension de ce résultat était la notion de jeu [40]. Cette idée est elle même le sujet de nombreuses ramifications.

Autres formes de langages. D'autres formes de langages ont été considérées comme les langages de traces (c'est à dire de mots modulo une commutation partielle de lettres [69]) ou d'images (c'est à dire de mots bidimensionnels indexés par un rectangle), ou les langages avec données (c'est à dire sur un alphabet infini, autorisant la comparaison de lettres [49]).

Dans ces différentes directions, toutes les facettes du théorème fondamental des langages réguliers ont été considérées. Certains résultats résistent, d'autres non. Par exemple, la notion d'automate déterministe passe aux mots infinis indexés par les ordinaux, mais n'a plus de sens pour des ordres plus généraux. La notion pose également problème pour les arbres infinis pour lesquels aucune forme d'automates déterministe équivalent aux langages réguliers n'est envisageable [20]. La description algébrique devient également problématique dans le cas arborescent. Nous n'allons pas plus avant dans cette description dans cette introduction.

Une autre branche très riche de la théorie des automates considère les questions de caractérisations de classes. Nous avons vu que, déjà sur les mots finis, la logique du premier-ordre n'est pas aussi expressive que la logique monadique, c'est à dire qu'il existe des langages réguliers qui ne peuvent être définis au moyen d'une formule en logique du premier ordre. Il s'agit en particulier du langage $(aa)^*$, c'est à dire le langage des mots de longueur paire. La question naturelle est donc de déterminer, parmi les langages réguliers, ceux qui peuvent être définis au moyen d'une formule du premier ordre. Le théorème de Schützenberger, McNaughton et Papert caractérise de manière effective cette classe.

Théorème 1.3 ([93, 72]). *Il est équivalent pour un langage de mots finis :*

- *d'être descriptible par une expression sans étoile (c.-à-d., utilisant le mot vide, les lettres, la concaténation, l'union et le complément),*

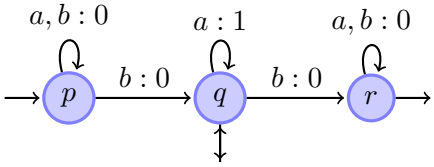
- *d'être définissable par une formule de la logique du premier ordre,*
- *d'être reconnu par un monoïde apériodique (c'est à dire qu'il existe n tel que $a^n = a^{n+1}$ pour tout a),*
- *d'avoir un monoïde syntactique apériodique (le monoïde syntactique est le plus petit qui reconnait le langage, et il peut être calculé),*
- *d'être accepté par un automate sans compteur (nous ne définisons pas cette notion ici),*
- *d'avoir un automate déterministe minimal sans compteur.*

De plus ces équivalences sont effectives et il existe un algorithme, qui étant donné un langage régulier décide s'il est définissable en logique du premier-ordre et, le cas échéant, renvoie une formule le définissant.

D'autres caractérisations de cette classe ont été décrites dans la littérature, comme au moyen de la logique temporelle LTL [50], et de nombreux autres résultats de caractérisation sont maintenant connus. Là encore, une description plus poussée de ce thème nous éloignerait trop du sujet précis de ce travail.

1.2 Hauteur d'étoile, automates de distance et variations

Une autre branche de la théorie des automates, moins classique, est à l'origine de la théorie des fonctions régulières de coût. Il s'agit des automates de distance. Les automates de distance sont des automates dont les transitions sont pondérées par des entiers :



Un tel automate n'est pas utilisé pour accepter un langage, mais plutôt pour accepter une fonction, des mots dans $\mathbb{N} \cup \{\infty\}$. Plus précisément, l'automate associe à chaque mot u :

> le minimum sur toutes les exécutions menant d'un état initial à un état final sur le mot u, de la somme des poids le long du chemin, ou ∞ s'il n'existe pas de telle exécution.

Ainsi l'automate ci-dessus, à chaque mot de la forme $a^{n_0}ba^{n_1}\dots ba^{n_k}$, associe la valeur $\min(n_1, \dots, n_k)$. Le lecteur averti reconnaitra dans ces automates de distance des automates pondérés (comme introduits par Schützenberger [92]) sur le semi-anneau tropical (*c.-à-d.* $(\mathbb{N}\cup\{\infty\}, \min, +)$). Il s'agit d'automates qui calculent des valeurs dans $\mathbb{N}\cup\{\infty\}$ en effectuant un produit le long des chemins (le «produit» est en fait $+$ dans le semi-anneau tropical), et en interprétant le choix non-déterministe comme une somme (la «somme» est en fait min dans le semi-anneau tropical).

Si les automates pondérés ont de nombreuses belles propriétés par eux-mêmes, ce n'est pas la raison pour laquelle Hashiguchi–dont les travaux vont nous intéresser–les a considérés. Hashiguchi s'intéressait au problème de la hauteur d'étoile (restreinte), posé par Eggan [37]. Ce problème s'énonce comme suit :

Entrée : Un langage régulier de mots L et une entier positif k.
Sortie : Oui s'il existe une expression rationnelle [2] qui s'évalue en L et utilise au plus k imbrications d'étoiles de Kleene, non autrement.

Eggan établit que cette hiérarchie est stricte, c'est à dire qu'il existe des langages exprimables au moyen de $k+1$ imbrications d'étoiles de Kleene qui sont impossibles à décrire avec seulement k imbrications d'étoiles. Le problème de la hauteur d'étoile est le problème de décision associé. Cette question a été rapidement considérée comme centrale en théorie des langages, et également comme l'une des plus difficiles du domaine.

Bien que certains résultats partiels aient été établis par McNaughton, Dejean et Schützenberger [71, 35], ce n'est que vingt-cinq ans plus tard que le problème a été résolu par Hashiguchi.

Théorème 1.4 ([43, 42, 44, 45])**.** *Le problème de la hauteur d'étoile est décidable.*

Quatre papiers sont nécessaires à l'établissement de cette preuve, qui est notoirement difficile. Hashiguchi, pour mener à bien sa démonstration réduit le problème de la hauteur d'étoile à un **problème d'existence de bornes** [3] :

Entrée : Un automate de distance.
Sortie : Oui s'il existe un entier n tel que sur tout mot la valeur calculé par l'automate n'excède pas n, non autrement.

Hashiguchi montre la décidabilité de ce problème :

Théorème 1.5 ([42])**.** *Le problème de l'existence d'une borne est décidable (pour les automates de distance sur les mots finis).*

La compréhension de ce résultat de décidabilité a par la suite été le sujet de nombreux travaux, en particulier de Leung et Simon. Simon a d'abord montré un résultat plus faible [96]. En s'inspirant de ces travaux Leung donne alors dans sa thèse une nouvelle preuve du résultat de Hashiguchi [63, 64, 65] en introduisant l'opération de stabilisation (que nous aurons l'occasion de présenter longuement dans ce document). Simon unifia ensuite ces résultats, en utilisant en particulier des arbres de factorisation [98] (qui peuvent s'interpréter

2. Une expression rationnelle est formée de lettres, de sommes, de produits et d'étoile de Kleene. Le problème de la hauteur d'étoile généralisée autorise l'utilisation de négations, et son statut est très différent : nul ne sait s'il existe des langages qui ne soient pas de hauteur 1.

3. La réduction se fait plus précisément au problème de «limitedness». Les deux problèmes sont équivalents en termes de décidabilité.

comme les prémices de certains outils décrits dans ce document). La complexité précise de ce problème est connue de Leung et Podolskiy [66], il est PSPACE-complet.

Pourtant, malgré ces nombreux travaux qui ont permis de bien comprendre le théorème 1.5, le reste de la preuve de Hashiguchi (c'est à dire la plus grande partie) restait encore très mal comprise. En effet, la réduction du problème de la hauteur d'étoile au problème d'existence de bornes, est extrêmement compliquée. En 2005, Kirtsten propose une preuve très différente de la décidabilité de la hauteur d'étoile [52]. Cette dernière est plus concise et plus informative sur la nature réelle du problème. Il s'agit, comme dans la preuve de Hashiguchi, d'une réduction à un problème d'existence de bornes. L'apport majeur de Kirsten a été de ne pas utiliser un automate de distance, mais les automates de «**distance-désert imbriqués**» qui en sont une extension stricte. Comme ces automates sont plus généraux, il est plus facile de réduire le problème de la hauteur d'étoile à l'existence de bornes pour ces automates. Nous verrons dans le cours de ce document les automates de «distance-désert imbriqués» sous le nom de «B-automates hiérarchiques». Bien entendu, Kirsten établit le résultat de décidabilité correspondant :

Théorème 1.6 ([52]). *Le problème de l'existence de bornes est décidable pour les automates de distance-désert imbriqués (appelés* hB*-automates dans la suite). Le problème est PSPACE-complet.*

Avec Christof Löding, nous avons poursuivi ce travail et étudié le cas des arbres. Une notion similaire à la hauteur d'étoile existe assez naturellement sur les arbres. En effet, il existe des expressions rationnelles qui permettent de dénoter les langages réguliers d'arbres, et elles coïncident sur les mots avec les expressions rationnelles sur les mots.

Théorème 1.7 ([31]). *Le problème de la hauteur d'étoile sur les arbres est décidable.*

Là preuve poursuit l'approche de Hashiguchi et Kirsten et s'obtient par réduction à un problème d'existence de bornes, cette fois-ci sur les arbres.

Théorème 1.8 ([31]). *Le problème de l'existence de bornes est décidable pour les* B*-automates hiérarchiques alternants sur les arbres finis.*

Cette dernière partie de la preuve nécessite de mettre en place des résultats de jeux concernant ces automates. Les prolongements de ces techniques seront longuement abordés dans la suite de ce document.

1.3 La logique MSO + $\mathbb{U}$

L'introduction des fonctions régulières de coût a été fortement motivée par l'étude d'une nouvelle logique étendant la logique monadique, la logique MSO+$\mathbb{U}$. Cette section peut être lue sous deux angles. D'une part, historiquement, elle correspond à des travaux antérieurs fondateurs des fonctions de coût. D'autre part la décidabilité de cette logique étant encore

ouverte, de nombreuses continuations possibles à donner aux fonctions régulières de coût sont motivées par l'étude de la logique MSO + $\mathbb{U}$.

La logique MSO + $\mathbb{U}$ s'obtient en partant de la logique monadique et en autorisant la construction $\mathbb{U}X\ \Phi$ signifiant [4] :

«il existe des ensembles arbitrairement grands tels que Φ est vrai»

ce qui s'abrège en «$\forall n\ \exists X\ |X| > n \wedge \Phi$». Remarquons que cette extension de la logique, bien entendu, n'est vraiment pertinente que sur les structures infinies. En effet, sur une structure finie, il suffit de remplacer toutes les occurrences de la nouvelle construction $\mathbb{U}X\ \Phi$ par faux pour obtenir une formule équivalente, de la logique monadique.

Bojańczyk a considéré cette extension afin de résoudre le problème de décider de l'existence d'un modèle fini pour une formule du μ-calcul avec modalité de retour [5]. Il montre qu'un fragment très restreint de cette logique est décidable sur les arbres infinis [4] (c'est à dire qu'il est possible de décider s'il existe un arbre infini qui satisfait cette formule). Dans ce fragment, la nouvelle construction ne peut apparaître (essentiellement) qu'à l'extérieur de la formule.

Nous avons ensuite essayé de résoudre complètement cette logique sur les mots infinis, sans succès. Nous ne sommes parvenus qu'à décider de la théorie sur les mots infinis pour un fragment restreint de la logique (fragment beaucoup plus général que celui considéré dans [4]).

Théorème 1.9 ([7]). *La théorie* MSO+$\mathbb{U}$ *est décidable sur les mots infinis pour les formules des fragments suivants :*

Fragment $\omega\mathtt{S}$ *: les formules dans lesquelles la construction* $\mathbb{U}$ *n'apparait que positivement dans la formule, i.e., sous un nombre pair de négation,*

Fragment $\mathcal{B}(\omega\mathtt{S})$ *: les combinaisons booléennes de formules de* $\omega\mathtt{S}$*,*

Fragment $\omega\mathtt{BS}$ *: les formules obtenues du fragment* $\mathcal{B}(\omega\mathtt{S})$ *par quantification existentielle, disjonction et conjonction.*

Remarquons que les langages définissables dans le fragment $\omega\mathtt{BS}$ ne sont pas clos sous complément (cela requiert une preuve), et donc ce fragment n'est pas aussi expressif que la logique MSO + $\mathbb{U}$ dans son entier.

Il est possible de déduire du théorème 1.9, par une réduction simple, les résultats de décidabilité pour les problèmes d'existence de bornes (les théorèmes 1.5 et 1.6). Par contre l'algorithme obtenu ainsi est non-élémentaire, alors que la bonne borne de complexité devrait être PSPACE. Les $\mathtt{B}$-automates et les $\mathtt{S}$-automates développés dans ce travail sont aussi issus de la preuve du théorème 1.9 (avec des définitions améliorées).

4. Cette logique est parfois aussi nommée de MSO + $\mathbb{B}$, dans laquelle la construction $\mathbb{B}X\ \Phi$ signifie qu'il existe une borne à la taille des ensembles tels que Φ est satisfaite. À négation près, les constructions $\mathbb{B}$ et $\mathbb{U}$ sont équivalentes. La construction $\mathbb{B}$ à le défaut d'être «anti-monotone» : en effet, plus «Φ est vraie», moins «$\mathbb{B}X\ \Phi$» est vraie. Nous jugeons cette propriété nuisible au développement propre de cette théorie. La construction $\mathbb{U}$ n'a pas ce défaut.

5. Bien qu'à postériori, ce problème soit plus du ressort des fonctions régulières de coût.

Pour l'instant aucune raison n'est entrevue permettant de conclure que cette logique serait indécidable.

Conjecture 1.10. *La logique* MSO + $\mathbb{U}$ *est décidable sur les mots infinis et les arbres infinis.*

Pour l'instant, le problème de résoudre la logique MSO + $\mathbb{U}$ semble hors de portée. Bojańczyk et Toruńczyk ont donc étudié la variante faible de cette logique, appelée WMSO+ $\mathbb{U}$. Dans cette logique, les variables monadiques ne peuvent prendre comme valeur que des ensembles finis.

Théorème 1.11 ([6, 10])**.** *La satisfaction de la logique* WMSO + $\mathbb{U}$ *est décidable sur les mots infinis et les arbres infinis*[6]*.*

En fait, ce que nous enseigne la preuve du théorème 1.11 sur les mots infinis est que toute formule de WMSO + $\mathbb{U}$ peut être traduite en une formule du fragment ωBS de la logique MSO + $\mathbb{U}$. Combiné avec le fait que les langages définissables en logique WMSO + $\mathbb{U}$ sont clos sous complément alors que les langages définissables en logique MSO + $\mathbb{U}$ ne le sont pas, il s'ensuit que MSO + $\mathbb{U}$ est strictement plus expressif que WMSO + $\mathbb{U}$ (cela diffère de la logique monadique classique pour laquelle MSO et WMSO coïncident sur les mots infinis).

La relation exacte entre les fonctions régulières de coût et la logique MSO + $\mathbb{U}$ n'est pas aisées à décrire. Il semble que décider la théorie de MSO + $\mathbb{U}$ sur les mots infinis (conjecture MSO + $\mathbb{U}$) soit un problème extrêmement compliqué qui met en jeu des phénomènes qui n'ont aucun équivalent dans la théorie des fonctions régulières de coût. En ce sens la logique MSO + $\mathbb{U}$ est plus complexe que les fonctions régulières de coût. D'un autre coté, tous les résultats connus de décidabilité de fragments de MSO + $\mathbb{U}$ s'obtiennent par réduction à des questions de fonctions régulières de coût. C'est le cas pour le théorème 1.9[7]. La preuve du théorème 1.11 utilise quant à elle explicitement une réduction à un résultat de décidabilité pour les fonctions de coût régulières sur les arbres finis, mais cette utilisation n'est pas essentielle : la preuve du théorème 1.11 n'utilise qu'un fragment très faible des fonctions régulières de coût, et nécessite par ailleurs de nombreux autres arguments qui n'ont pas d'équivalents dans la théorie des fonctions régulières de coût.

Certaines pistes ayant pour objectif de comprendre finement la relation des fonctions régulières de coût et la logique MSO + $\mathbb{U}$ et ces variantes seront présentées au cours du chapitre 11.

6. C'est aussi le cas de la logique WMSO + $\mathbb{R}$ sur les mots infinis [9], où $\mathbb{R}$ est un autre opérateur qui est tel que MSO + $\mathbb{R}$ à la même expressivité que MSO + $\mathbb{U}$, mais WMSO + $\mathbb{R}$ et WMSO + $\mathbb{U}$ sont d'expressivités incomparables.

7. Le preuve originale n'utilise pas les fonctions régulières de coût, mais pourrait être drastiquement simplifiée en faisant usage des théorèmes présentés dans ce travail.

Séparations topologiques.

Avant de clore cette section sur la logique MSO+$\mathbb{U}$, il est bon de préciser que les relations entre les divers fragments connus décidables de la logique MSO+$\mathbb{U}$ ont été finement étudiées sous l'angle topologique. En effet la hiérarchie des boréliens et la hiérarchie projective, qui sont issues de la théorie descriptive des ensembles, proposent des notions pertinentes pour étudier les langages de mots infinis.

La hiérarchie de Borel.

Les **Boréliens** d'un espace topologique sont des parties de cet espace considérée comme «topologiquement simples». Elles s'obtiennent à partir des ouverts et des fermés par clôture sous union dénombrable et intersection dénombrable. La **hiérarchie de Borel** raffine cette définition en associant à chaque Borélien un ordinal. Il permet de préciser l'instant d'apparition de de chaque ensemble au cours de l'itération de point fixe ci-dessus. Ainsi, étant donnée une famille d'ensembles X, notons X_σ sa clôture sous union dénombrable et X_δ sous intersection dénombrable. La hiérarchie de Borel commence au niveau 1 en définissant Σ^0_1 comme l'ensemble des ouverts, et Π^0_1 comme l'ensemble des fermés. Les ensembles Σ^0_α et Π^0_α pour un ordinal α sont alors définis par induction comme :

$$\Sigma^1_\alpha = (\bigcup_{\beta<\alpha} \Pi^0_\beta)_\sigma \ , \qquad \text{et} \quad \Pi^1_\alpha = (\bigcup_{\beta<\alpha} \Sigma^0_\beta)_\delta \ ,$$

En particulier, cela signifie pour les niveaux entiers que

$$\Sigma^1_{n+1} = (\Pi^0_n)_\sigma \ , \qquad \text{et} \quad \Pi^1_{n+1} = (\Sigma^0_n)_\delta \ .$$

Bien entendu, cette hiérarchie est croissante : seuls Σ^0_α et Π^0_α sont susceptibles d'être incomparables. On peut aussi remarquer que les ensembles de Σ^0_α sont exactement les complémentaires des ensembles de Π^0_α. Tout aussi naturellement, tous les ensembles créés ainsi sont Boréliens. En fait, tout Borélien est atteint à un ordre dénombrable au cours de cette induction (cela vient du fait que les deux opérations permettant la construction de cette hiérarchie, l'union dénombrable et l'intersection dénombrable, sont d'arité dénombrable). Ainsi, rien ne sert de poursuivre cette induction au delà de des ordinaux dénombrables : aucun nouvel ensemble ne serait produit.

Une propriété simple, mais importante, de la hiérarchie de Borel est que l'image inverse par une fonction continue d'un ensemble de Σ^0_α (resp. Π^0_α) est dans Σ^0_α (resp. Π^0_α). Cela vient du fait que l'image inverse d'un ouvert (resp. d'un fermé) par une fonction continue est un ouvert (resp. un fermé), et que l'image inverse d'une union est l'union des images inverses (ainsi que pour l'intersection). Pour cette raison, un ensemble X est dit complet pour Σ^0_α (resp. Π^0_α) si tout ensemble de Σ^0_α (resp. Π^0_α) s'obtient par image inverse par une fonction continue de X. Cette notion d'ensemble complet est à la base de la plupart des techniques de séparation entre les niveaux des classes de la hiérarchie de Borel.

Cette hiérarchie s'instancie dans différents espaces topologiques. Nous nous intéresserons à l'espace de Cantor. Il s'agit pour nous de l'ensembles des mots sur un alphabet fini donné, équipé de la topologie induite par la distance $d(u,v) = 2^{-|u \sqcap v|}$ où $u \sqcap v$ est le plus long préfixe commun à u et v (et $d(u,u) = 0$ bien entendu). Pour cette topologie, la hiérarchie est stricte : tous les niveaux indexés par des ordinaux dénombrables sont distincts. Les ensembles ouverts de cette topologie sont décrits par un ensemble de mots finis F, et l'ensemble ouvert correspondant contient tous les mots infinis ayant un mot de F comme préfixe. Les ensembles fermés sont décrits au moyen d'un ensemble de mots finis P, et l'ensemble fermés correspondant est l'ensemble des mots infinis dont tous les préfixes sont dans P. Les ensembles qui sont à la fois ouverts et fermés pour cette topologie sont ceux pour lesquels il existe un entier n tel que pour déterminer si $u \in X$ il suffit de connaitre les n premières lettres du mot. Ainsi, l'ensemble des mots qui portent la lettre a à la position i est ouvert et fermé. En effet, il suffit de connaitre les i premières lettres du mot pour établir ou réfuter cette propriété.

La hiérarchie de Borel reflète naturellement des phénomènes logiques. Si l'on considère par exemple une définition logique de la propriété «il y a une infinité d'occurrences de la lettre a» comme

$$\forall i \in \mathbb{N}\ \exists j \in \mathbb{N}\ j > i \wedge a(j)\ ,$$

alors il est possible d'en déduire une borne supérieure sur le niveau dans la hiérarchie de Borel de cet ensemble, comme suit. La quantification $\forall i \in \mathbb{N}$ est vue comme une intersection dénombrable, et $\exists j \in \mathbb{N}$ comme une union dénombrable. On remarque de plus que $j > i \wedge a(j)$ est une propriété ouverte et fermée (en effet, pour i et j fixés, il suffit de considérer le mot jusqu'à la position $\max(i,j)$ pour savoir s'il satisfait la formule). Il s'ensuit que la propriété «il y a une infinité d'occurrences de la lettre a» s'exprime comme une intersection dénombrable d'unions dénombrables d'ouverts. Ainsi, cet ensemble appartient à Π^0_2 (et en fait, il est complet pour ce niveau).

Ce type d'argument se généralise à toute logique dont les quantifications portent sur des variables prenant leurs valeurs dans un ensemble dénombrable. Ces logiques ne peuvent que définir des ensembles appartenant aux ω premiers niveaux de la hiérarchie de Borel. C'est le cas de WMSO ou de WMSO + $\mathbb{U}$.

En revanche, les quantifications sur les ensembles (et plus généralement sur les relations) permettent de sortir de la classe des Boréliens. On atteint alors la hiérarchie projective.

La hiérarchie projective.

Nous avons utilisé ci-dessus le fait que les quantifications prennant leurs valeurs sur des ensembles dénombrables se modélisent naturellement par des intersections ou des unions dénombrables. Cet argument n'est plus valable pour les quantifications monadiques. L'opération correspondante est la projection, ce qui dans ce contexte signifie l'image directe par une application continue. En effet, imaginons une formule de la forme $\exists X\ \Phi$. Alors

Φ peut être vue comme définissant un ensemble de mots infinis sur l'alphabet $\mathbb{A} \times \{0,1\}$, dans lequel le bit supplémentaire représente la fonction caractéristique de l'ensemble de X. Ainsi, Φ définit un ensemble de paires (mot, ensemble), ou l'ensemble est représenté comme sa fonction caractéristique. La quantification existentielle s'interprète alors comme l'opération éliminant la composante supplémentaire représentant la valeur courante de X. Il s'agit d'une projection, et cette application est continue dans la topologie que nous avons présentée ci-dessus (elle est 1-lipschitzienne).

L'opération de projection et l'opération d'image par une fonction continue font en général sortir de la hiérarchie Borélienne. Il suffit même de prendre l'image d'un ouvert par une fonction continue. Un ensemble est **analytique** s'il est l'image par une fonction continue d'un Borélien. L'ensemble des ensembles analytiques est dénoté Σ^1_1. Le complémentaire d'un ensemble analytique est dit **co-analytique**. Les ensembles co-analytiques sont dénotés Π^1_1. De manière générale, la hiérarchie projective est définie comme suit : pour tout entier n, Π^1_n contient les complémentaires des ensembles de Σ^1_n, et Σ^1_{n+1} contient les images par fonctions continues des ensembles dans Π^1_n. Dans la topologie de Cantor, la hiérarchie projective est stricte.

Résultats topologiques concernant la logique MSO + $\mathbb{U}$.

Maintenant que les notions de théorie descriptive des ensembles adéquates ont été présentées, nous pouvons énoncer les résultats connus concernant la complexité topologique de différents fragments de MSO + $\mathbb{U}$.

Il est bien connu que les langages réguliers sur les mots infinis sont des combinaisons booléennes de langages de Σ^0_2 (en effet, par le théorème de McNaughton [70], les langages réguliers d'ω-mots s'écrivent comme des combinaisons booléennes de langages Büchi déterministes, et ces derniers sont, par définition, Π^0_2), et n'appartiennent pas, en général, aux classes inférieures de la hiérarchie de Borel. Une analyse similaire a été menée dans le cadre de la logique MSO + $\mathbb{U}$:

Théorème 1.12 ([18, 48]). *Les énoncés suivants concernent les langage d'ω-mots :*

- *Tout langage définissable en* WMSO + $\mathbb{U}$ *est une combinaison booléenne de langages* Σ^0_2.
- *Les langages définissables en* MSO *sont strictement*[8] *inclus dans les langages définissables en* WMSO + $\mathbb{U}$.
- *Les langages définissables dans le fragment* ωS *de* MSO + $\mathbb{U}$ *sont dans* Π^0_3 *et il en existe qui sont complets pour* Π^0_3.
- *Les langages définissables dans le fragment* ωB *de* MSO + $\mathbb{U}$ *sont dans* Σ^0_4 *et il en existe qui sont complets pour* Σ^0_4.
- *Les langages définissable en logique* MSO + $\mathbb{U}$ *sont dans la hiérarchie projective et il en existe qui sont complets pour chaque niveau de la hiérarchie projective.*

8. Ce résultat n'est pas obtenu par un argument topologique puisque WMSO + $\mathbb{U}$ et MSO rencontrent les mêmes niveaux de la hiérarchie de Borel.

Nous voyons en particulier que les fragments de MSO$+\mathbb{U}$ décidables par le théorème 1.9 sont encore très loin de la complexité (topologique) de la logique MSO$+\mathbb{U}$ complète. Bien entendu, ces résultats ne signifient en rien par eux-mêmes que décider de la logique MSO$+\mathbb{U}$ est une question difficile. En effet, cette manière de classifier topologiquement les langages n'a pas de rapport, *a priori*, avec une quelconque effectivité. En revanche, cela donne des indications très importantes sur la nature des objets qu'il convient de développer si l'on cherche à résoudre la conjecture 1.10.

Ainsi, nous savons immédiatement qu'aucun modèle d'automate qui aurait une condition d'acceptation borélienne n'est à même de capturer les langages définissables en MSO$+\mathbb{U}$.

1.4 Les fonctions régulières de coût

Dans les sections précédentes, nous avons présenté la théorie classique des langages réguliers, et les automates de distance et leurs extensions.

> Est-il possible de décrire une théorie expliquant tous les résultats de décidabilité d'existence de bornes et qui serait en même temps une extension de la théorie des langages réguliers ? En particulier, existe-t-il une telle théorie dans laquelle le théorème fondamental des langages réguliers (théorème 1.1) est valide ?

L'obstacle principal à l'établissement d'une telle extension est que, dès que les automates peuvent manipuler des valeurs entières, même de manière faible comme les automates de distance, les problèmes deviennent très rapidement indécidables. C'est ce que nous enseigne le théorème suivant dû à Krob.

Théorème 1.13 (Krob [56])**.** *Déterminer si les fonctions calculées par deux automates de distance sont égales est un problème indécidable.*

En revanche l'équivalence entre deux langages réguliers est un problème décidable, et il est difficile d'envisager une généralisation de la notion de langage régulier qui ne préserve pas une telle propriété. La solution adoptée dans la théorie des fonctions régulières de coût est de ne considérer les fonctions que modulo une relation d'équivalence [28, 25].

Deux fonctions f, g de E (typiquement E est l'ensemble des mots sur un alphabet fini) dans $\mathbb{N} \cup \{\infty\}$ sont **$\approx$-équivalentes** si pour toute partie $X \subseteq E$,

f est bornée sur X si et seulement si g est bornée sur X.

Une *fonction de coût* (sur E) est une classe d'équivalence pour $\approx$.

Cette relation d'équivalence identifie de nombreuses fonctions, mais elle préserve pourtant les questions de bornes. En particulier, f est bornée si elle est $\approx$-équivalente à la fonction nulle. De plus, cette relation d'équivalence est compatible avec certaines opérations comme le minimum ou le maximum. Ainsi, si $f \approx f'$ et $g \approx g'$ alors $\min(f, g) \approx \min(f', g')$ et $\max(f, g) \approx \max(f', g')$.

Quand les fonctions sont considérées modulo $\approx$, c'est à dire quand les fonctions de coût sont utilisées, et non les fonctions usuelles, alors il est possible de retrouver un théorème fondamental.

Théorème 1.14 ([25, 28]). *Les propriétés suivantes sont équivalentes pour une fonction de coût sur les mots finis :*

- *être définissable en logique monadique de coût,*
- *être accepté par un* B*-automate [hiérarchique]*[9]*,*
- *être accepté par un* B*-automate déterministe en histoire*[10] *[hiérarchique],*
- *être accepté par un* S*-automate,*
- *être accepté par un* S*-automate déterministe en histoire,*
- *être décrit par une* B*-expression régulière,*
- *être décrit par une* S*-expression régulière,*
- *être reconnu par un monoïde de stabilisation.*

Ces équivalences sont effectives et l'égalité entre fonctions de coût de cette classe est décidable[11].

Assez naturellement, les fonctions de coût acceptées par l'une des classes ci-dessus sont appelées **régulières**.

Les objets ci-dessus seront présentés dans la suite de ce document. Les B-automates, en particulier, sont des variantes des automates de distance-désert de Kirsten [52] qui ont été introduits dans [7]. Ils étendent les automates de distance que nous avons déjà rencontrés.

Il est important de noter les points suivants :

– Il s'agit d'une extension stricte du théorème fondamental sur les langages réguliers. En effet, un langage peut être vu comme une fonction associant la valeur 0 aux mots du langage, et la valeur ∞ aux mots en dehors du langage. En ce sens, le théorème étend le théorème fondamental des langages réguliers. Cette extension est stricte, en effet (a) tout langage régulier est représenté en ce sens par une fonction régulière de coût, et (b) il existe des fonctions régulières de coût qui ne correspondent à aucun langage.

– Les résultats de décidabilité de l'existence d'une borne de Hashiguchi et Kirsten (théorèmes 1.5 et 1.6) se déduisent du théorème ci-dessus. En effet, une fonction est bornée si et seulement si elle est $\approx$-équivalente à la fonction constante 0. Ceci est décidable d'après le théorème 1.14.

9. Les automates hiérarchiques utilisent des conditions d'acceptations plus contraintes, comme le sont les conditions de parité par rapport aux conditions de Streett.

10. Cette notion remplace le déterminisme. Les B-automates déterministes, eux, sont strictement plus faibles.

11. En fait la relation de domination est décidable. Nous dirons que f domine g ($f \succcurlyeq g$ dans la suite) si pour tout sous-ensemble $X \subseteq E$, si $g(X)$ est borné, alors $f(X)$ l'est aussi. Dans le cas présent, les deux résultats de décidabilité sont équivalents. En effet, $f \approx g$ si $f \preccurlyeq g$ et $g \preccurlyeq f$. Réciproquement, $f \preccurlyeq g$ si et seulement si $\min(f, g) \approx f$, et les fonctions de coût de cette classe sont closes sous min.

– Les résultats plus complexes comme la décidabilité de l'existence de bornes pour la logique monadique de coût sont nouveaux et ne peuvent être déduits des résultats de Hashiguchi et Kirsten.

– Cette approche permet de contourner le résultat d'indécidabilité de Krob (théorème 1.13) puisque l'équivalence de deux fonctions pour $\approx$ est, elle, décidable.

Nous ne présentons pas ici tous les objets impliqués dans le théorème 1.14, puisque ce document est en grande partie consacré à leur description.

Néanmoins, afin de donner plus de corps à la description précédente, introduisons tout de même l'un des objets impliqués dans le théorème fondamental des fonctions de coût, la **logique monadique de coût**. Un exemple de formule de la logique monadique est la formule `segment` suivante :

$$\texttt{segment}(X) ::= \\ (\forall y \in X\ a(y)) \wedge (\forall x, y, z (x \in X \wedge z \in X \wedge x < y < z) \rightarrow y \in X)\ .$$

La formule énonce en langage courant, que «X est un ensemble de positions consécutives qui portent toutes la lettre a». La logique **monadique de coût** autorise alors de construire la formule suivante :

$$\forall X\ \texttt{segment}(X) \rightarrow |X| \leq N\ .$$

Cette formule est vraie sur un mot u si tous les segments de a consécutifs ont la taille au plus N. Cette formule est en fait vue comme une fonction, la fonction qui à chaque mot associe le plus petit entier N rendant la formule vraie. Dans le cas présent, la formule définit la fonction qui à $a^{n_0}ba^{n_1}b\dots ba^{n_k}$ associe la valeur $\max(n_0, n_1, \dots, n_k)$.

Il s'agit du point de départ de la théorie des fonctions régulières de coût. Nous verrons dans le reste de ce travail, entre autre, que :

– Le théorème 1.14 s'étend aux mots infinis.

– Le théorème 1.14 s'étend aux arbres finis (sauf la notion de déterminisme en histoire et la notion de monoïde qui n'ont plus de sens) [33].

– Le théorème 1.14 s'étend aux arbres infinis quand seule la logique monadique de coût faible[12] est considérée [12] (les automates correspondants sont alors des `B`/`S`-automates alternants faibles).

– La logique monadique de coût faible et la logique monadique de coût coïncident effectivement sur les mots infinis [13].

– Il est possible de caractériser effectivement certaines classes de fonctions de coût sur les mots finis [30, 58].

– La conjecture la plus importante dans ce cadre est que le théorème 1.14 s'étend aux arbres infinis.

12. Rappelons qu'il s'agit du fragment dans lequel seules les quantifications sur les ensembles finis sont autorisées.

1.5 L'approche profinie de Toruńczyk

Au cours de sa thèse [102], Szymon Toruńczyk a développé une approche plus abstraite permettant d'aborder la logique MSO + $\mathbb{U}$ et en particulier les fonctions régulières de coût. L'idée est la suivante :

> Si l'on pouvait traiter les fonction de coût comme des ensembles (par exemple, de mots) plutôt que des fonctions, alors l'approche serait plus uniforme avec la théorie des ensembles.

C'est ce que propose l'approche profinie. En effet, s'il faut voir les fonctions comme des ensembles, alors, quels sont les éléments de ces ensembles ? La réponse apportée par Toruńczyk est que des ensembles de mots profinis peuvent jouer ce rôle.

Les mots profinis sont des classes d'équivalence de suites de mots finis. Une suite de mots finis est **convergente** si pour tout langage régulier L, les mots de la séquence sont tous ultimement dans L, ou bien ils sont tous ultimement hors de L. Dans le premier cas, nous dirons que la séquence appartient à L, dans le cas contraire qu'elle appartient à $\complement L$. Deux suites de mots finis convergentes sont **équivalentes** si elles appartiennent exactement aux même langages réguliers. Un mot profini est une classe d'équivalence de suites convergentes pour cette relation d'équivalence. L'ensemble des mots profinis sur l'alphabet $\mathbb{A}$ est noté $\widehat{\mathbb{A}^*}$. Étant donné un langage régulier $L \subseteq \mathbb{A}^*$, nous notons $\hat{L}$ l'ensemble des mots profinis qui lui appartiennent, c'est à dire les suites convergentes ultimement dans L. Les mots profinis sont munis d'une topologie (en fait, une ultramétrique). Les ouverts de bases sont les $\hat{L}$ pour L langage régulier. Dans cette topologie, l'application produit est continue. L'opération ω qui à $u_1, \dots$ associe $u_1^{1!}, u_2^{2!}, u_3^{3!} \dots$ est également une application continue.

Pour montrer la relation avec le travail présent, il s'agit de montrer comment les langages de mots profinis sont en relation avec les fonctions de coût. Soit une fonction $f : \mathbb{A}^* \to \mathbb{N} \cup \{\infty\}$, il est possible de lui associer l'ensemble de mots profinis (*c.-à-d.* de suites convergentes) $\hat{f} \subseteq \widehat{\mathbb{A}^*}$ suivant :

$$\hat{f} = \{\vec{u} \in \widehat{\mathbb{A}^*} \ : \ \limsup_i f(u_i) < \infty\} = \bigcup_{n \in \mathbb{N}} \{\vec{u} \ : \ f(u_i) \leq n, \text{ pour tout } i\} \ .$$

Fort de cette traduction, toute fonction peut être vue comme un ensemble (ouvert) de mots profinis. Une forme de réciproque est possible. Tout ensemble ouvert de mots profinis reconnu (dans un sens qui ne sera pas développé dans ce document) correspond à une fonction de coût régulière.

Ainsi, il est possible d'identifier les objets utilisés dans la théorie des fonctions de coût. Dans ce cadre il est possible d'énoncer tous les résultats sur les mots de ce travail. En particulier, les notions de monoïdes, d'automates, et de logique peuvent s'interpréter dans le cadre profini. La relation entre ces deux travaux est néanmoins assez complexe à décrire, et nous ne rentrons pas dans les détails.

Nous verrons au cours du chapitre 11 une autre approche, grandement inspirée des travaux de Toruńczyk, et à la fois très différente, permettant de s'abstraire d'une partie de la

technique des fonctions régulières de coût. Cette nouvelle approche utilise, à la place des mots profinis, l'existence de modèles non-standards des mathématiques (de ZFC) en provenance de l'analyse non-standard. De l'avis de l'auteur cette approche possède de meilleurs propriétés que les mots profinis. Il est bien entendu très prématuré d'effectuer une telle comparaison.

1.6 Quelques problèmes qui sont reliés aux fonctions régulières de coût

Il se trouve qu'un certain nombre de questions de la littérature s'avèrent reliées directement ou indirectement aux questions d'existence de bornes et aux fonctions régulières de coût. Nous essayons d'en donner une liste, non-exhaustive dans cette section.

La puissance finie. Étant donné un langage régulier L, il s'agit de décider s'il existe un entier n tel que

$$L^* = (L + \varepsilon)^n \ .$$

Cette question est très naturelle, puisqu'elle revient à déterminer si l'étoile de Kleene doit nécessairement être itérée une infinité de fois avant d'atteindre le point fixe. Ce problème a été montré décidable par Simon [96] et Hashiguchi [41] indépendamment. La preuve de Hashiguchi est une réduction directe aux problèmes d'existence de bornes pour les automates de distance.

La hauteur d'étoile restreinte. Ce problème a déjà été présenté en détail au cours de la section 1.2. Le cas de la hauteur d'étoile sur les arbres finis est lui aussi décidable (voir section 8.1).

Le problème de la substitution finie. Il s'agit, étant donnés deux langages réguliers $L \subseteq \mathbb{A}^*$ et $K \subseteq \mathbb{B}^*$, de déterminer s'il existe une substitution $\sigma\ \mathbb{A} \to \mathcal{P}(\mathbb{B}^*)$ finie (*c.-à-d.*, telle que $\sigma(a)$ est fini pour toute lettre a) telle que $\sigma(L) = K$. Ce résultat a été établi indépendamment par Bala et Kirsten, par une réduction à un problème d'existence de bornes pour les «automates de désert» [1, 51].

La hauteur d'étoile relative. Il s'agit d'une variante de la question de la hauteur d'étoile qui peut être résolue par la même approche [45, 53].

La hiérarchie de Mostowski sur les arbres infinis. Sur les arbres infinis, les automates utilisent une condition d'acceptation, comme la condition de parité ou la condition de Rabin. Ces conditions sont paramétrées, par le nombre de paires de Rabin (pour la condition de Rabin), ou le nombre de priorités distinctes (pour la condition de parité). Ce paramètre est important, par exemple, dans les procédures de décision. Il induit également une hiérarchie sur les langages. Il s'agit de la hiérarchie de Rabin-Mostowski. Le problème correspondant consiste, étant donné un langage régulier d'arbres infinis, à déterminer à quel niveau il se trouve dans cette hiérarchie. Ce problème se réduit à un problème d'existence de bornes pour des fonctions représentées par automates sur les arbres infinis [32]. Ce dernier est toujours ouvert. Ces questions seront le sujet de la section 9.2.

L'itération bornée de point-fixes. Une formule $\Phi(X, y)$ (en logique monadique), avec comme paramètre un ensemble X et un élément y, est **positive en** X si tous les tests d'appartenance $z \in X$ utilisés dans la formule (pour une variable du premier-ordre z) apparaissent positivement (*i.e.*, sous un nombre pair de négations). Une telle formule peut être vue, sur chaque structure relationnelle, comme une application des ensembles d'éléments dans les ensembles d'éléments :

$$F_\Phi^{\mathcal{S}}(E) = \{v \ : \ \mathcal{S} \models \Phi(E, v)\} \ .$$

L'hypothèse de monotonicité en X sur Φ garantit que cette fonction est monotone pour l'inclusion. Elle possède donc un plus petit point fixe, $\mathrm{Fix}_\Phi^{\mathcal{S}}$. La question de l'itération bornée consiste à déterminer si ce point fixe peut être obtenu en partant de $\emptyset$ par application d'un nombre fini de fois la fonction $F_\Phi^{\mathcal{S}}$:

> Étant donnée une formule formule $\Phi(X, y)$ positive en x, existe t'il un entier n tel que pour toute structure relationnelle $\mathcal{S}$,
>
> $$\mathrm{Fix}_\Phi^{\mathcal{S}} = (F_\Phi^{\mathcal{S}})^n(\emptyset) \ .$$

Ce problème est décidable sur les mots finis [2], et sur les arbres infinis [3].

Variantes quantitatives du problème de synthèse de Church. Étant donné un langage régulier de mots infinis L, considérons le jeu dans lequel les joueurs `I` et `II` alternativement écrivent un bit, jusqu'à produire un mot infini. Si le mot produit est dans L, le joueur `I` gagne. Le problème de synthèse de Church consiste à déterminer quel joueur gagne, et si c'est le cas pour le premier joueur, à **synthétiser** un **contrôleur** pour le joueur, *c.-à-d.* à produire une machine déterministe qui lit en entrée la suite des bits, et décrit la stratégie que `I` doit suivre afin de gagner. Büchi et Landweber ont montré que ce problème est décidable et qu'il suffit d'utiliser comme contrôleur des automates à état fini [17].
Récemment une variante de cette question a été posée par Rabinovich et Velner [103]. Le problème est de déterminer s'il existe un entier n tel que, après avoir joué une partie infinie, le joueur `I` a la possibilité de modifier jusqu'à n des bits qu'il a joués, afin de tomber dans L. Ainsi, il s'agit de déterminer s'il existe n tel que `I` gagne le jeu L_n, où L_n est l'ensemble des mots qui ne diffèrent d'un mot de L que par au plus n lettres à des positions impaires. Cette question se traduit directement dans la théorie des fonctions de coût régulières. Le problème est, là encore, décidable, et un automate à état fini suffit comme contrôleur pour le joueur `I` (non-publié).

Extensions de la logique monadique. La logique MSO + $\mathbb{U}$ est à l'origine de la théorie des fonctions de coût. En fait, tous les résultats de décidabilité connus concernant la logique MSO + $\mathbb{U}$ se démontrent en utilisant la théorie des fonctions régulières de coût et plus particulièrement le théorème fondamental correspondant. Ainsi, décider du «fragment ωBS» (voir théorème 1.9), bien que la preuve originale soit différente, s'avère un résultat beaucoup plus aisé en utilisant l'équivalence effective entre `B`-automates et `S`-automates.

Chapitre 2

Logique monadique de coût et fonctions de coût

Le sujet de cette thèse est la description de la notion de fonctions régulières de coût. Ce chapitre présente l'approche logique aux fonctions régulières de coût, c'est à dire la logique monadique de coût. Il s'agit du modèle commun reliant le cas des mots finis, infinis, ainsi que des arbres finis et infinis. En ce sens, ce formalisme est central, contrairement aux notions d'algèbres, d'automates, ou d'expressions régulières développées par la suite, qui sont spécifiques à l'étude de modèles particuliers.

Il n'est pas question dans ce chapitre de décrire de procédures de décision (autres que triviales) concernant la logique monadique de coût. Ces techniques seront développées, pour les mots, dans la partie II, et pour les arbres, dans la partie III.

Ce chapitre se décompose comme suit. La section 2.1 donne un bref aperçu de la théorie standard concernant la logique monadique usuelle, ainsi que quelques résultats clef s'y rapportant. La section 2.2 introduit la logique monadique de coût, objet central de ce chapitre, et plus généralement de ce travail. La section 2.3 a pour but de brièvement énoncer les résultats de décidabilité ou d'indécidabilité directement hérités de la littérature. La section 2.4 présente les problèmes centraux que l'on cherche à résoudre. L'équivalence de borne et les fonctions de coût, notions également centrales, sont introduites à la section 2.5. Enfin, la section 2.6 présente une approche générique pour décider de la logique monadique de coût, qui sera instanciée dans la suite du document.

2.1 La logique monadique

Rappelons tout d'abord que la **logique monadique (du second-ordre)** (ou plus simplement la **logique monadique** dans la suite) est l'extension de la logique du premier ordre autorisant l'utilisation de variables d'ensemble (dites variables monadiques). Concrètement les formules de la logique monadique utilisent des variables du premier ordre (notées

par convention en minuscule $x, y, \dots$) et des variables monadiques (notées par convention en capitale $X, Y, \dots$). Il est possible dans une formule monadique d'utiliser des quantifications existentielles et universelles sur les variables du premier ordre comme sur les variables monadiques, d'utiliser tous les connecteurs logiques usuels, d'exprimer l'appartenance d'un élément à un ensemble ($x \in Y$), ainsi que de toutes les relations de la structure.

Exemple 2.1. Un exemple typique d'usage de la logique monadique est d'exprimer l'existence d'un chemin dans un graphe (orienté). Dans ce cas, les éléments de la structure sont les sommets du graphe, et l'unique relation $\texttt{arc}(x, y)$ exprime l'existence d'une arête d'origine x et de destination y. L'existence d'un chemin menant d'un sommet x vers un sommet y peut s'énoncer comme le fait que tout ensemble contenant x et clos par la relation `arc` contient aussi y. Ceci se formalise en logique ainsi :

$$\texttt{chemin}(x, y, X) ::= \forall Z \\ \big(x \in Z \wedge \forall z, z' \ (z \in Z \wedge z' \in X \wedge \mathbf{arc}(z, z') \to z' \in Z)\big) \quad \to \quad y \in Z$$

Cette formule énonce une propriété un peu plus précise : elle requiert, étant donné un sommet initial x, un sommet final y, et un ensemble de sommets X, que le chemin menant de x à y n'emprunte que des arcs aboutissant dans X.

Rappelons également brièvement quelques définitions que nous utiliserons par la suite. Une **signature** est un ensemble de **symboles** $R, S, \dots$. Dans la suite, les signatures seront toujours implicites, et ne comportent qu'un nombre fini de symboles. À chaque symbole est associé un entier naturel appelé son **arité**, également implicite dans la suite. Une **structure** (relationnelle) $\mathcal{S} = \langle U_\mathcal{S}, R^\mathcal{S}, \dots \rangle$ (sur une signature donnée $R, \dots$) est composée d'un ensemble $U_\mathcal{S}$ appelé son **univers** et d'une relation $R^\mathcal{S} \subseteq U_\mathcal{S}^k$ pour chaque symbole R d'arité k de la signature. La relation $R^\mathcal{S}$ est appelée **interprétation** du symbole R dans $\mathcal{S}$.

Étant donné un ensemble de **variables** V, partagé en variables du premier ordre et variables monadiques, une **valuation de** F (sur $\mathcal{S}$) est une application associant à chaque variable du premier ordre x un élément de $U_\mathcal{S}$ et à chaque variable monadique une partie de $U_\mathcal{S}$. On notera par $\mathrm{v}, X = E$ (*resp.* $\mathrm{v}, x = e$) la valuation de $F \cup \{X\}$ (*resp.* de $F \cup \{x\}$) coïncidant avec v sur $F \setminus \{X\}$ (*resp.* sur $F \setminus \{X\}$) et qui à X associe E (*resp.* et qui à x associe e).

Étant données une formule φ et une valuation v de ses variables libres sur une structure $\mathcal{S}$, on notera $\mathcal{S}, \mathrm{v} \models \varphi$ pour signifier que la formule φ est satisfaite sur $\mathcal{S}$ quand ses variables libres prennent les valeurs données par v. Dans le cas où il n'y a aucune variable libre, on notera simplement $\mathcal{S} \models \varphi$, et on dira que $\mathcal{S}$ est un **modèle de** φ.

Concernant la logique monadique, la plupart des propriétés (non-triviales) sont indécidables. C'est en particulier le cas du problème de satisfaction : décider si une formule de logique monadique admet un modèle est indécidable (c'est déjà le cas pour la logique du premier-ordre).

Pour cette raison, la logique monadique est en général étudiée sur une classe $\mathcal{C}$ de structures fixée. Les classes les plus étudiées sont les suivantes :

– les mots finis,
– les mots de longueur ω (et parfois au delà de ω),
– les arbres finis,
– les arbres infinis,
– les structures de largeur arborescente/largeur de clique/largeur de partition bornée.

Dans tous ces cas, l'existence d'un modèle pour une formule donnée dans l'une de ces classes est un problème décidable, et bien compris. Le dernier cas se dérive en fait de cas des arbres. En effet, les structures de largeur de arborescente/de clique/de partition bornée sont toutes interprétables dans les arbres infinis. Cette remarque (que nous ne développerons pas plus) procure une réduction au cas des arbres. En ce sens les résultats développés dans ce document restent valables sur les structures de largeur arborescente bornée.

Dans la suite, nous ne considérerons que les mots et les arbres (finis ou infinis).

Les mots. Chaque **mot fini** sur un alphabet $\mathbb{A}$ est vu comme une structure relationnelle dont l'univers est $\{1, \ldots, n\}$, où n est la longueur du mot, équipée d'une relation binaire $<$ qui coïncide avec l'ordre usuel sur $\{1, \ldots, n\}$, et pour chaque lettre $a \in \mathbb{A}$, le symbole a est interprété comme l'ensemble des position portant la lettre a. L'ensemble des mots finis sur l'alphabet $\mathbb{A}$ est noté $\mathbb{A}^*$.
Les **mots infinis** (aussi appelés **ω-mots** dans la suite) sur un alphabet $\mathbb{A}$ sont définis de manière similaire, à la différence près que l'univers est $\mathbb{N}$. L'ensemble des mots infinis sur l'alphabet $\mathbb{A}$ est noté $\mathbb{A}^\omega$.

Les arbres. Les **arbres** sont définis sur un alphabet gradué. Un **alphabet gradué** $\mathbb{A}$ est un ensemble de symboles équipés d'une **arité** entière. Un arbre sur l'alphabet $\mathbb{A}$ est un une fonction $T : \mathbb{N}^\omega \to \mathbb{A}$ telle que :
– Le domaine de T est clos par préfixe. Ses éléments sont appelés **nœuds**. On écrira simplement $u \in T$ au lieu de $u \in dom(T)$.
– Si $u \in T$, et $T(u)$ est d'arité k, alors $ui \in T$ si et seulement si $i \in \{1, \ldots, k\}$. Le nœud ui est appelé le $i$ème **fils** de u. Le nœud u est le **parent** de ui.

Les nœuds u tels que $T(u)$ est d'arité 0 sont appelés **feuilles**. Le nœud ε est appelé **racine**. Un arbre est dit **fini** si son domaine est fini. Par opposition, on parlera d'**arbre infini** si l'ensemble de nœuds est fini ou infini (le terme infini est utilisé par opposition au cas fini).
Les arbres sont aussi vus comme des structures relationnelles. La structure correspondante un arbre T est telle que :
– l'univers est l'ensemble des nœuds, $dom(T)$,
– pour chaque lettre a d'arité k, il existe un symbole correspondant a dans la signature, d'arité $k+1$. L'interprétation de ce symbole est :

$$a^T \stackrel{\texttt{def}}{=} \{(u, u1, u2, \ldots, uk) \ : \ T(u) = a\} \ .$$

Dans la suite, l'une de ces classes sera toujours fixée. On parlera d'un **langage** de mots (*resp.* de mots infinis, d'arbres, ou d'arbres infinis) pour signifier un ensemble de mots (*resp.* de mots infinis, d'arbres, ou d'arbres infinis). On dira qu'un langage est **définissable en**

logique monadique s'il existe une formule de la logique monadique qui a pour modèles exactement les éléments de la classe considérée.

À titre d'exemple, la propriété

$$\exists x\, \exists y\ a(x) \wedge b(y) \wedge \forall z\, (x < z \wedge z < y) \rightarrow c(y)$$

est satisfaite exactement par les mots du langage rationnel $\mathbb{A}^* ac^* b\mathbb{A}^*$.

Cette relation entre langages rationnels et logique monadique est générale comme précisé par le fameux théorème de Büchi-Elgot-Trakhtenbrot-Rabin (une partie du théorème fondamental 1.1) :

Théorème 2.2 (Büchi,Elgot,Trakhtenbrot,Rabin)**.** *Un langage de mots finis (resp. de mots de longueur ω, d'arbres finis, ou d'arbres infinis) sur un alphabet fini fixé est définissable en logique monadique si et seulement s'il est accepté par automate fini de mots finis (resp. automate de Büchi, automate d'arbres, automate d'arbres infinis). De plus, ces équivalences sont effectives.*

Bien entendu, cet énoncé n'est pas complètement spécifié tant que les définitions adéquates d'automates n'ont pas été données. Chacune de ces définitions sera introduite au moment opportun.

2.2 Logique monadique de coût

La logique monadique de coût étend la syntaxe de la logique monadique par l'usage d'une unique variable N d'une nouvelle nature ; N est une **variable de borne** qui prend des valeurs entières. La syntaxe de la **logique monadique de coût** est exactement celle de la logique monadique, augmentée d'un unique nouveau prédicat $|X| \leq N$ (appelé **prédicat de cardinal**), où X est une variable monadique et N la variable de borne, avec la contrainte que ce nouveau prédicat apparaisse **positivement** dans chaque formule, *c.-à-d.* sous un nombre pair de négations. Comme on peut s'y attendre, $|X| \leq N$ énonce que le cardinal de l'ensemble X est inférieur à N.

Étant donnée une formule φ (de la logique monadique de coût), on dénote par $\mathrm{FV}(\varphi)$ ses variables libres (variable de borne exclue). Une formule sans variable libre est appelée **phrase**. Étant donnés une structure $\mathcal{S}$, une formule monadique de coût φ, une valuation v de ses variables libres, et un entier n (positif), on dénote par

$$\mathcal{S}, \mathrm{v}, n \models \varphi$$

le fait que la structure satisfait la formule φ pour la valuation des variables v et la valuation n de la variable de borne .

La contrainte de positivité a une conséquence directe.

Fait 2.3. Pour tous entiers $m \geq n$, toute formule monadique de coût φ, toute structure relationnelle $\mathcal{S}$, et toute valuation v,

$$\mathcal{S}, \mathrm{v}, n \models \varphi \qquad \text{implique} \qquad \mathcal{S}, \mathrm{v}, m \models \varphi \ .$$

Les formules de la logique monadique de coût ne sont en réalité pas utilisées pour déterminer une valeur booléenne, mais plutôt pour calculer une valeur. Ainsi, une formule de la logique monadique de coût φ de variables libres F sert à représenter une fonction des structures dans $\mathbb{N} \cup \{\infty\}$. Formellement, pour toute structure $\mathcal{S}$, et toute valuation v des variables libres de φ, on pose :

$$[\![\varphi]\!](\mathcal{S}, \mathrm{v}) \stackrel{\texttt{def}}{=} \inf\{n \ : \ \mathcal{S}, \mathrm{v}, n \models \varphi\} \ .$$

Cette valeur est entière s'il existe une valeur de n telle que φ est satisfaite. Dans le cas contraire, $[\![\varphi]\!](\mathcal{S}, \mathrm{v})$ vaut ∞ (c'est la raison pour laquelle inf est utilisé, et non min). Dans le cas d'une phrase, on omet naturellement la valuation, et on écrit simplement $[\![\varphi]\!](\mathcal{S})$.

Il est d'ores et déjà possible de souligner la relation que la logique monadique de coût entretient avec la logique monadique usuelle :

Fait 2.4. Pour toute formule monadique φ, toute valuation v et toute structure relationnelle $\mathcal{S}$,

$$[\![\varphi]\!](\mathcal{S}, v) = \begin{cases} 0 & \text{si } \mathcal{S}, v \models \varphi \\ \infty & \text{sinon.} \end{cases}$$

Exemple 2.5. La phrase $\forall X \ |X| \leq N$ calcule la taille de la structure, *c.-à-d.* $[\![\forall X \ |X| \leq N]\!](\mathcal{S})$ vaut $|U_{\mathcal{S}}|$, le nombre d'éléments de la structure.

Un exemple plus intéressant prolonge l'exemple 2.1. Considérons, de nouveau utilisant la signature d'un di-graphe, la formule :

$$\texttt{diamètre} ::= \forall x, y \ \exists X \ |X| \leq N \wedge \texttt{chemin}(\texttt{x}, \texttt{y}, \texttt{X}) \ .$$

Elle définit le diamètre du graphe, en effet, le diamètre d'un graphe est le plus petit entier n tel qu'entre toute paire de sommets il existe un chemin du premier au second sommet n'empruntant que des arcs terminant dans un ensemble de sommets de cardinal au plus n.

Afin de se prémunir contre certaines considérations non cruciales, nous fixons pour la suite plus précisément la syntaxe de la logique monadique de coût comme suit :

$$\begin{aligned} \varphi ::= \quad & \exists X \ \varphi \quad | \quad \forall X \ \varphi \quad | \quad \varphi \wedge \psi \quad | \quad \varphi \vee \psi \\ | \quad & R(X_1, \ldots, X_k) \quad | \quad \neg R(X_1, \ldots, X_k) \quad | \quad |X| \leq N \end{aligned}$$

où $X, X_1, \ldots, X_k$ sont des variables monadiques, et R est un symbole d'arité k, qui peut éventuellement être $\subseteq$. Remarquons en particulier que les prédicats de symbole usuels apparaissent en forme positive et négative, alors que le prédicat $|X| \leq N$, lui, n'apparaît que positivement dans cette syntaxe.

Ce type de syntaxe est classique, et correspond à garantir les points suivants :

a. seules les variables monadiques sont autorisées (il n'y a donc pas de variables du premier-ordre, et par conséquent aucune apparition du prédicat d'appartenance $\in$, qui est remplacé par $\subseteq$),

b. il est possible d'utiliser la relation d'inclusion entre ensembles $X \subseteq Y$,

c. les relations de la structure sont interprétées comme des relations sur les ensembles singletons correspondant, et

d. les négations n'apparaissent qu'au niveau des prédicats.

Tout comme dans le cas classique, il est tout aussi simple dans le cas de la logique monadique de coût d'établir que toute formule peut être transformée en une formule équivalente respectant cette syntaxe particulière. Pour cela il suffit de remplacer les variables du premier-ordre par des ensembles singletons (être singleton se définit dans la logique comme être minimal pour l'inclusion parmi les ensembles non-vides). Pousser les négations aux feuilles est également une opération standard, dans laquelle en particulier on utilise les équivalences entre $\neg\exists X\ \varphi$ et $\forall X\ \neg\varphi$, ainsi qu'entre $\neg\forall X\ \varphi$ et $\exists X\ \neg\varphi$ et les lois de De Morgan. C'est aussi pour cette raison que l'on ne peut pas omettre dans cette syntaxe les quantificateurs universels.

Nous avons vu comment la sémantique $[\![\varphi]\!]$ d'une formule se décrivait en terme de la relation $\models$. Il est également possible de donner une définition inductive directe de $[\![\varphi]\!]$ comme suit.

Fait 2.6. Étant donnés une structure $\mathcal{S}$ et une valuation v, les égalités suivantes sont satisfaites :

$$[\![R(X_1,\dots,X_k)]\!](\mathcal{S},\mathrm{v}) = \begin{cases} 0 & \text{si } R^{\mathcal{S}}(\mathrm{v}(X_1),\dots,\mathrm{v}(X_k)) \\ \infty & \text{sinon} \end{cases}$$

$$[\![\neg R(X_1,\dots,X_k)]\!](\mathcal{S},\mathrm{v}) = \begin{cases} \infty & \text{si } R^{\mathcal{S}}(\mathrm{v}(X_1),\dots,\mathrm{v}(X_k)) \\ 0 & \text{sinon} \end{cases}$$

$$[\![|X| \leq N]\!](\mathcal{S},\mathrm{v}) = |\mathrm{v}(X)|$$
$$[\![\varphi \vee \psi]\!](\mathcal{S},\mathrm{v}) = \min([\![\varphi]\!](\mathcal{S},\mathrm{v}), [\![\psi]\!](\mathcal{S},\mathrm{v}))$$
$$[\![\varphi \wedge \psi]\!](\mathcal{S},\mathrm{v}) = \max([\![\varphi]\!](\mathcal{S},\mathrm{v}), [\![\psi]\!](\mathcal{S},\mathrm{v}))$$
$$[\![\exists X\ \varphi]\!](\mathcal{S},\mathrm{v}) = \inf\{[\![\varphi]\!](\mathcal{S},\mathrm{v}, X = E)\ :\ E \subseteq U_{\mathcal{S}}\}$$
$$[\![\forall X\ \varphi]\!](\mathcal{S},\mathrm{v}) = \sup\{[\![\varphi]\!](\mathcal{S},\mathrm{v}, X = E)\ :\ E \subseteq U_{\mathcal{S}}\}$$

En appliquant ces égalités par récurrence sur la formule, il est possible de directement définir $[\![\varphi]\!]$.

À la lumière de ces relations, la logique monadiqe de coût peut être vue comme une extension de la logique standard, dans laquelle la quantification existentielle est remplacée par un *infimum*, et la quantification universelle par un *supremum* travaillant sur $\mathbb{N} \cup \{\infty\}$.

2.3 Cas simples de décidabilité et d'indécidabilité

La logique monadique de coût est fortement reliée à la logique monadique classique. Pour cette raison, certaines questions sont décidables simplement par réduction à la logique monadique. Cette section sera aussi l'occasion de rappeler le théorème d'indécidabilité de Krob. L'ensemble de ces énoncés délimite la zone qui est intéressante d'investiguer concernant la logique monadique de coût. Les questions simples s'obtiennent comme simples conséquences de la théorie classique, et les questions trop ambitieuses sont rapidement indécidables.

Proposition 2.7. *Pour toute formule de logique monadique de coût φ, et tout $k \in \mathbb{N}$, il existe une formule de logique monadique ψ telle que pour toute structure :*

$$[\![\varphi]\!](\mathcal{S}) \leq k \qquad \textit{si et seulement si} \qquad \mathcal{S} \models \psi \ .$$

Démonstration. Il suffit de remarquer que pour tout k, l'énoncé $|X| \leq k$ peut s'exprimer en logique monadique (par exemple par un formule signifiant «pour tous $x_0 \ldots, x_k$, dans X, il existe $i \neq j$ tels que $x_i = x_j$»). La formule ψ s'obtient alors de la formule φ en substituant à toutes les occurrences d'un prédicat $|X| \leq N$ la formule correspondant à $|X| \leq k$. Remarquons que cette preuve ne fait pas usage de le propriété de positivité dans l'apparition des prédicats $|X| \leq n$, elle serait tout aussi juste sans cette contrainte. CQFD

Il s'ensuit que toute propriété de la forme $[\![\varphi]\!](\mathcal{S}) \leq k$ est décidable sur toutes les classes pour lesquelles la logique monadique est décidable.

En revanche, décider s'il existe une structure $\mathcal{S}$ telle que $[\![\varphi]\!](\mathcal{S}) = \infty$ est un problème un peu plus délicat. On appelle le **support** d'une fonction à valeur dans $\mathbb{N} \cup \{\infty\}$ l'ensemble des éléments envoyés par la fonction sur une valeur finie. L'énoncé suivant montre qu'il est aisé de définir les structures *finies* appartenant au support d'une fonction définie par une formule monadique de coût.

Proposition 2.8. *Pour toute formule de logique monadique de coût φ, il existe une formule de logique monadique ψ telle que pour toute structure finie :*

$$[\![\varphi]\!](\mathcal{S}) < \infty \qquad \textit{si et seulement si} \qquad \mathcal{S} \models \psi \ .$$

Démonstration. La formule ψ s'obtient en remplaçant dans φ tout prédicat de cardinal $|X| \leq N$ par vrai.

Il est trivial que φ implique ψ, en effet la formule «vrai» est toujours plus vraie que la formule $|X| \leq N$ (quelle que soit la valuation de N), ce qui s'étend à la formule globale en utilisant la contrainte de positivité dans l'utilisation des prédicats $|X| \leq N$.

Réciproquement, si $\mathcal{S} \models \psi$, alors $\mathcal{S}, |\mathcal{U}_\mathcal{S}| \models \varphi$. En effet, toute partie de l'univers de $\mathcal{S}$ a une taille bornée par $|\mathcal{U}_\mathcal{S}|$, ainsi, $|X| \leq |\mathcal{U}_\mathcal{S}|$ est équivalent à vrai sur la structure $\mathcal{S}$. CQFD

Remarque 2.9. En revanche, cette propriété est fausse si la structure est infinie. Un exemple sur les mots infinis sur l'alphabet $\{a, b\}$ est donné par la fonction suivante :

$$f(u) = \begin{cases} 0 & \text{si } u \text{ contient un nombre fini d'occurrences de } b \\ \sup\{n_i \ : \ i \in \mathbb{N}\} & \text{si } u = a^{n_1}ba^{n_2}b\cdots \end{cases}$$

Cette fonction f se définit aisément en logique monadique de coût par une formule énonçant que, «soit il y a un nombre fini d'occurrences de b, soit tous les segments de a consécutifs ont une taille bornée par N». En revanche, le langage $L = \{u \ : \ f(u) < \infty\}$ n'est pas régulier. En effet, le langage L contient tous les mots ultimement périodiques, et le seul langage régulier à posséder cette propriété est $\{a, b\}^\omega$ (un langage régulier de mots infinis est entièrement caractérisé par les mots ultimement qu'il contient, voir [16]). Or L est différent de $\{a, b\}^\omega$ car il ne contient pas le mot $a^1ba^2ba^3b\ldots$ Ainsi, L n'est pas régulier, ce qui prouve que la proposition 2.8 ne s'étend pas aux mots infinis.

Les propriétés que nous avons vues ci-dessus relient la logique monadique de coût à la logique monadique. Les problèmes qui n'ont pas de contrepartie dans le cadre classique sont plus intéressants à étudier. Malheureusement, l'indécidabilité arrive rapidement comme le montre le résultat de Krob (théorème 1.13 dans l'introduction) :

Théorème 2.10 (Krob [56])**.** *Le problème de l'égalité des fonctions définissables en logique monadique de coût est indécidable sur les mots. L'indécidabilité s'obtient déjà pour les formules «***de distance***», c.-à-d. de la forme* $\exists X \ |X| \leq N \wedge \psi$*, ou* ψ *est monadique*[1]*.*

2.4 Les problèmes centraux

Les propriétés auxquelles nous nous intéressons dans l'étude de la logique monadique de coût sont pleinement inspirées de la branche de théorie dont elle sont issues ([42, 65, 96, 52]). La question originale est celle de l'existence d'une borne. D'autres problèmes reliés, comme la divergence ou la domination sont au centre de cette étude. Nous nous intéressons aux propriétés suivantes. Ces problèmes sont paramètres par une classe de structures fixées, comme les mots ou les arbres.

Existence d'une borne Il existe un entier n tel que $[\![\varphi]\!](\mathcal{S}) \leq n$ pour toute structure de $\mathcal{C}$.

Divergence Pour tout n, il n'existe qu'un nombre fini de structures $\mathcal{S}$ de $\mathcal{C}$ telles que $[\![\varphi]\!](\mathcal{S}) \leq n$,

Domination Pour tout sous-ensemble $E \subseteq \mathcal{C}$, si $[\![\varphi]\!]$ est bornée sur E alors $[\![\psi]\!]$ est aussi bornée sur E. On dit dans ce cas que φ **domine** ψ.

1. L'énoncé original fait état de fonctions calculées par des automates de distance. En anticipant quelque peu, nous utilisons ici la proposition 5.15 qui énonce que les fonctions calculées par les automates de distance sont celles correspondant au fragment logique mentionné ci-dessus.

Ces questions ne peuvent être réduites (en tous cas aisément) à la logique monadique usuelle. De plus, ces questions deviennent indécidable si l'hypothèse de positivité dans l'usage des prédicats de cardinal est retirée. L'objectif des techniques développées dans ce travail est de résoudre ces problèmes d'existence de borne, de divergence, et de domination, ainsi que de voir la conséquence de ces résultats.

Il est aisé de voir que le problème de domination est une extension à la fois du problème d'existence de borne ainsi que du problème de divergence. En effet, vrai (rappelons que $[\![\text{vrai}]\!]$ est la fonction constante égale à 0) domine ψ si et seulement si ψ est bornée. De même φ domine $\forall X\ |X| \leq N$ (rappelons que $[\![\forall X\ |X| \leq N]\!](\mathcal{S})$ est le cardinal de l'univers de $\mathcal{S}$) si et seulement si φ est divergente (ici on utilise que le nombre de structures d'un cardinal fini donné est fini, *c.-à-d*, que la signature est finie).

Dans le cas de formules monadiques φ et ψ, grâce au fait 2.4, on obtient que φ domine ψ si et seulement si ψ implique φ (et comme cas particulier, $[\![\varphi]\!]$ est bornée si et seulement si φ est une tautologie (sur $\mathcal{C}$), et $[\![\varphi]\!]$ est divergente si et seulement si φ n'est satisfaite que sur un nombre fini de structures).

Historiquement, à l'initiative de Hashiguchi, la notion de **fonction limitée** est utilisée [42]. Elle revient à décider si une fonction est bornée sur son support. Nous avons vu à la remarque 2.9 que la notion de support ne se comporte pas très bien dans le cas des objets infinis. Pour cette raison, nous préférons éviter cette notion dans la suite de ce document.

Au cours de la section suivante, nous introduisons la notion de fonction de coût qui est l'objet d'étude remplaçant la notion de langage de la théorie classique.

2.5 Domination, équivalence et fonctions de coût

Nous avons vu la relation de domination comme un problème à résoudre à la section précédente. Nous reprenons cette notion dans cette section, pour en faire l'objet central d'étude. En effet, afin de contourner le résultat d'indécidabilité de du théorème 2.10, la solution adoptée est de considérer toutes les fonctions modulo la relation d'équivalence correspondant à la domination mutuelle.

Dans la suite, on dira qu'une fonction à valeurs dans $\mathbb{N}\cup\{\infty\}$ est **bornée** sur X s'il existe un entier n tel que $f(x) \leq n$ pour tous les x dans X. Remarquons une fois de plus qu'il y a deux raisons pour une fonction f de ne pas être bornée sur un ensemble X : soit $f(x) = \infty$ pour un certain $x \in X$, soit il existe une suite $(x_i)_{i\in\mathbb{N}} \in X^{\mathbb{N}}$ telle que $(f(x_i))_i \to \infty$.

Définition 2.11. Soit E un ensemble, et f, g deux applications de E dans $\mathbb{N} \cup \{\infty\}$, alors f est **dominée** par g, ce que l'on note $f \preccurlyeq g$, si pour tout sous-ensemble $X \subseteq E$,

$$g \text{ bornée sur } X \qquad \text{implique} \qquad f \text{ bornée sur } X\ .$$

L'équivalence correspondante $f \approx g$ correspond naturellement à $f \preccurlyeq g$ et $g \preccurlyeq f$. On appelle **fonction de coût** (sur E) une classes d'équivalence pour $\approx$.

Exemple 2.12. Sur les mots, $|\cdot|_a \preccurlyeq |\cdot|$ (où $|\cdot|_a$ compte le nombre d'occurrences de la lettre a). En effet, sur tout sous-ensemble de mots de longueur bornée, le nombre d'occurrence de a est aussi borné. En revanche $|\cdot|_a \not\preccurlyeq |\cdot|_b$, car sur l'ensemble $X = a^*$, $|\cdot|_b$ est bornée, alors que $|\cdot|_a$ ne l'est pas.
Similairement, les fonctions $|\cdot|_a$ et $|\cdot|_b$ sont incomparables pour la domination. En effet $X = a^*$ est un témoins que $|\cdot|_b$ ne domine pas $|\cdot|_a$, et réciproquement, $X = b^*$ est un témoins que $|\cdot|_a$ ne domine pas $|\cdot|_b$.
Remarquons aussi que si $f \leq g$ alors $f \preccurlyeq g$.

Pour montrer que des fonctions sont comparables pour $\preccurlyeq$, il est utile d'utiliser un raffinement de $\preccurlyeq$.

Définition 2.13. On appelle **fonction de correction** ou **fonction d'étirement** une fonction croissante α de ω dans ω que l'on étend à $\mathbb{N} \cup \{\infty\}$ en posant $\alpha(\infty) = \infty$. On pose alors :

$$f \preccurlyeq_\alpha g \qquad \text{si} \qquad f \leq \alpha \circ g \ ,$$

et $f \approx_\alpha g$ si $f \preccurlyeq_\alpha g$ et $g \preccurlyeq_\alpha f$.

L'idée est que $f \preccurlyeq_\alpha g$ énonce qu'une fois étirée par α, g est supérieure à f. De là provient la terminologie de fonction d'étirement utilisée pour nommer α. Une autre façon de voir est que pour $f \approx g$, f et g représentent la même fonction à une erreur près. La fonction α peut être vue comme «mesurant l'erreur à corriger» pour réconcilier f et g. Cela dit, on ne peut pas définir une distance au sens standard qui caractériserait l'erreur à corriger. Les bonnes notions mathématiques sont celles d'uniformité et de semi-uniformité qui sont moins naturelles à utiliser. Nous n'emploierons pas cette terminologie.

Les relations $\preccurlyeq$ est $\preccurlyeq_\alpha$ sont reliées par la proposition suivante.

Proposition 2.14. *Les propriétés suivantes sont équivalentes :*

1. *il existe α tel que $f \preccurlyeq_\alpha g$,*
2. *$\forall n \ \exists m \ \forall x \in E \ g(x) \leq n \to f(x) \leq m$, et*
3. *$f \preccurlyeq g$.*

Démonstration. La relation entre les deux premiers items est une forme de «skolémisation». La relation entre les deux derniers items correspond simplement à une explicitation des bornes impliquées dans la définition de $\preccurlyeq$.

De 1 à 2. Supposons $f \preccurlyeq_\alpha g$ pour une certaine fonction de correction α. Soit n un entier, posons $m = \alpha(n)$, On a pour pour tout x, si $g(x) \leq n$ alors $f(x) \leq (\alpha \circ g)(x) \leq \alpha(n) = m$. Ainsi le second item est satisfait.

De 2 à 3. Soit X un sous-ensemble de E. Supposons que g est borné sur X. Soit donc n un majorant de g sur X. D'après le second item, il existe m tel que $\forall x \in E \ g(x) \leq n \to$

$f(x) \leq m$. En particulier, pour tout $x \in X$, on a $g(x) \leq n$, et donc $f(x) \leq m$. Ainsi, f est majorée par m sur X.

De 3 à 1. Pour tout entier n, considérons l'ensemble $X_n = \{x \ : \ g(x) \leq n\}$. La fonction g est majorée sur X_n par n, et par conséquent (item 3), f est aussi majorée par une entier. Posons $\alpha(n) = \sup X_n$ (un tel majorant). Montrons que $f \preccurlyeq_\alpha g$. Soit donc $x \in X$. Supposons $g(x) < \infty$, on a alors $x \in X_{g(x)}$ par définition des ensembles X_i. Il s'ensuit que $f(x) \leq \sup f(X_{g(x)}) = \alpha(g(x))$. Dans le cas $g(x) = \infty$, on a $f(x) \leq \alpha(g(x)) = \infty$. Au total, $f \preccurlyeq_\alpha g$. CQFD

Il est aussi important de remarquer que, contrairement à $\preccurlyeq$, la relation $\preccurlyeq_\alpha$ n'est pas transitive à α fixé. Le fait suivant énonce quelques propriétés satisfaites par les relations $\preccurlyeq_\alpha$:

Fait 2.15. Si $f \preccurlyeq_\alpha g$ alors $f \preccurlyeq_{\alpha'} g$ pour tout $\alpha' \geq \alpha$.
Si $f \preccurlyeq_\alpha g \preccurlyeq_{\alpha'} h$ alors $f \preccurlyeq_{\alpha' \circ \alpha} h$.

Exemple 2.16. On a $|\cdot|_a + |\cdot|_b \approx_{\times 2} \max(|\cdot|_a, |\cdot|_b)$, en effet :

$$\max(|\cdot|_a, |\cdot|_b) \leq |\cdot|_a + |\cdot|_b \leq 2\max(|\cdot|_a, |\cdot|_b) \ .$$

Ainsi, sur $E = \{a, b\}^*$, $|\cdot| = |\cdot|_a + |\cdot|_b \approx \max(|\cdot|_a + |\cdot|_b)$.

Sur $E = \{$ensemble des arbres finis de rang non fixé$\}$,

$$\mathit{taille} \approx_{n \mapsto n^n} \max(\mathit{degmax}, \mathit{hauteur})$$

où $\mathit{taille}(t)$ représente le nombre de nœuds de t, $\mathit{degmax}(t)$ est le degré maximal d'un nœud de t, et $\mathit{hauteur}(t)$ est la longueur de la plus longue branche de t. L'égalité ci-dessus s'établit en montrant que pour tout arbre de taille t de degré maximum d et de hauteur h on a :

$$\max(d, h) \leq t \leq d^h \leq \max(d, h)^{\max(d,h)} \ .$$

Cet énoncé (élémentaire) est une sorte de variante du lemme de König. Celui-ci énonce qu'un arbre est infini si et seulement si il contient un nœud de degré infini ou une branche infinie. L'équivalence ci-dessus peut se lire «un arbre a un grand nombre de nœuds si et seulement s'il contient soit un nœud de grand degré soit une longue branche.»

Afin de mettre en évidence la relation entre la logique monadique classique et la logique monadique de coût, le principal pont est le fait de pouvoir voir un langage comme une fonction de coût. Sa fonction caractéristique joue ce rôle.

Remarque 2.17. Pour tout ensemble E et tout $X \subseteq E$, considérons la **fonction caractéristique** de X (dans E) définie pour tout $x \in E$ par :

$$\chi_X(x) = \begin{cases} 0 & \text{si } c \in X \\ \infty & \text{sinon.} \end{cases}$$

On vérifie aisément que pour tout X, Y sous-ensembles de E, $\chi_X \preccurlyeq \chi_Y$ ssi $Y \subseteq X$.

Le reste de cette section est constitué de remarques aidant à gagner de l'intuition sur la notion de fonction de coût. Ces résultats ne sont nullement nécessaires pour la suite.

Remarque 2.18. Une autre façon d'aborder la notion de fonction de coût est de l'interpréter en termes d'idéaux. Plus précisément un **idéal** I sur les parties d'un ensemble (la définition plus générale se fait sur un ensemble ordonné) est un ensemble de parties qui est :

1. clos vers le bas, *c.-à-d*, tel que $A \subseteq B$ et $B \in I$ implique $A \in I$, et ;
2. clos par union finie, *c.-à-d*, pour tous $A \in I$ et $B \in I$, $A \cup B \in I$.

Étant donné une fonction f de E dans $\mathbb{N} \cup \{\infty\}$, on associe l'ensemble $I_f = \{A \subseteq E \ : \ f \text{ est bornée sur } A\}$. On vérifie aisément que I_f est un idéal, et de plus que $I_f \subseteq I_g$ si et seulement si f domine g.

Une forme de réciproque est également possible. Une **base** d'un idéal I est une partie B de I telle que tout ensemble A de I est inclus dans une union finie d'éléments de B. Dit différemment, I est le plus petit idéal contenant B. En particulier, l'ensemble $B \subseteq I_f$ défini comme $\{\{x \in E \ : \ f(x) \leq n\} \ : \ n \in \mathbb{N}\}$ est une base dénombrable de I_f. Réciproquement, soit I un idéal de E de base dénombrable, et $B = \{B_i \ : \ i \in \mathbb{N}\}$ une base dénombrable de I, alors on définit $f_B(x)$ pour tout $x \in E$ par :

$$f_B(x) = \inf\{n \ : \ n \in B_n\} \ .$$

Vérifier que $I = I_{f_B}$ est élémentaire.

Ainsi, il est possible d'identifier les fonctions de coût sur un ensemble donné aux idéaux de base dénombrable.

Poursuivons notre description des propriétés élémentaires de la relation $\approx$. Elle se comporte bien vis à vis de certaines constructions comme le montre le fait suivant.

Fait 2.19. Si $f \approx f'$ et $g \approx g'$ alors $\min(f, g) \approx \min(f', g')$ et $\max(f, g) \approx \max(f', g')$. Si $f_i \approx_\alpha g_i$ pour tout $i \in I$, alors $\inf_{i \in I} f_I \approx_\alpha \inf_{i \in I} g_I$. Idem pour sup.

Ce deuxième fait donne quelques éléments descriptifs supplémentaires concernant la structure des fonctions de coût.

Fait 2.20. L'ensemble des fonctions de E dans $\omega + 1$ quotienté par $\approx$ est un treillis. Il possède une plus petite classe : la classe des fonctions bornées. Il possède une plus grande classe : la classe de la fonction constante égale à ∞. Si E est dénombrable, le «sous-treillis» induit par les fonctions de E dans $\mathbb{N}$ possède un élément maximal (la classe de une/toutes les bijections de E sur $\mathbb{N}$).

Nous concluons en montrant que la notion de fonctions de coût raffine strictement la notion d'ensemble. En effet, il n'existe qu'une unique fonction de coût à valeur dans $\mathbb{N}$ sur un ensemble E qui soit la fonction caractéristique d'un ensemble. La proposition suivante montre qu'il existe beaucoup plus de fonctions de coût de ce type.

Proposition 2.21. *Pour E ensemble infini dénombrable, l'ensemble des fonctions de E dans $\mathbb{N}$ a le cardinal du continu.*

Démonstration. On pose, sans perte de généralité, $E = N \setminus \{0\}$. Soit $p_0, \dots$ la suite des nombres premiers. Tout nombre $n \in E$ s'écrit de manière unique comme $p_1^{n_1} p_2^{n_2} \dots$ (où presque tous les n_i sont nuls, le sens de ce produit infini étant évident). Pour tout I, on pose $f_I(n) = \max\{n_i \ : \ n = p_1^{n_1} \dots, \ i \in I\}$. Soient $I \neq J$. On suppose, sans perte de généralité, qu'il existe $i \in I \setminus J$. Considérons la suite $y_n = p_i^n$. On a $f_I(y_n) = n$, et $f_J(y_n) = 0$. Ainsi, f_J est bornée sur $Y = \{y_n \ : \ n \in N\}$, alors que f_I ne l'est pas. Donc $f_I \not\approx f_J$. Comme ω possède cardinal du continu sous ensembles, on en déduit qu'il existe autant de classe d'equivalence pour la relation $\approx$. CQFD

2.6 Décider la domination sur une classe de structures

Nous avons présenté la logique monadique de coût à la Section 2.2 ainsi que les problèmes de décidabilité auxquels nous nous intéressons à la section 2.4 : l'existence de borne, la divergence, et la domination. Enfin, nous avons vu au cours de la section 2.5 la notion de fonction de coût, qui sera dorénavant la terminologie employée pour aborder ce type de questions.

Nous décrivons maintenant un cadre général permettant de décider les questions de domination sur une classe de structures donnée. Ce cadre sera ensuite appliqué sur les mots finis, les mots infinis, et les arbres finis et infinis (bien que dans ce dernier cas, la décidabilité reste une question ouverte). Nous nous fixons donc une classe de structures $\mathcal{C}$ qui peut être les mots, les mots infinis, les arbres, les arbres finis ou les arbres infinis.

Plus précisément, il est nécessaire de disposer d'une classe de structures qui nous permette de traiter les valuations des variables. Pour cela, on suppose que toute paire $(\mathcal{S}, \mathrm{v})$ formée d'une structure $\mathcal{S}$ de $\mathcal{C}$ et d'une valuation v sur $\mathcal{S}$ peut être vue comme une nouvelle structure $\mathcal{S}, \mathrm{v}$. On requiert que cette opération soit injective, et que la signature de $\mathcal{S}, \mathrm{v}$ ne dépende que de la signature de $\mathcal{S}$ et des variables dans le domaine de v. Si deux validations v, v' ont des domaines disjoints, on utilisera la notation v, v' la valuation unificatrice. Cela nous permet de noter indifféremment $\mathcal{S}, \mathrm{v}, \mathrm{v}'$ ou $\mathcal{S}, \mathrm{v}', \mathrm{v}$. On dira que la classe $\mathcal{C}$ est close sous **cylindrifiction**.

Sur la classe des fonctions de $\mathcal{C}$ dans $\mathbb{N} \cup \{\infty\}$, les opérations suivantes sont toujours définies, où f, g sont de telles fonctions :

Cylindrification. étant donné une structure $\mathcal{S}$ et une variable monadique X, la cylindrification de f est la fonction qui à $\langle \mathcal{S}, \mathrm{v} \rangle$ associe $f(\mathcal{S})$, où v et une valuation de la variable X,

Minimum et maximum se définissent naturellement, produisant $\min(f, g)$ et $\max(f, g)$.

Projections . Étant donnée une variable X, l'inf**-projection** et la sup**-projection** sont définies de f par rapport à X.

$$f_{\inf,X}(\mathcal{S}) = \inf\{f(\mathcal{S}, \mathrm{v}) \ : \ \mathrm{v} \text{ valuation de } X\} \ ,$$
$$\text{et} \quad f_{\sup,X}(\mathcal{S}) = \sup\{f(\mathcal{S}, \mathrm{v}) \ : \ \mathrm{v} \text{ valuation de } X\} \ .$$

Cardinal. Étant donnée une structure valuée $\mathcal{S}, \mathrm{v}$, et une variable X dans le domaine, on pose $\mathrm{card}_X(\mathcal{S}, \mathrm{v}) = |\mathrm{v}(X)|$.

Remarquons que toute ces définitions sont compatibles avec la notion de fonction de coût. Ainsi, si $f \approx f'$ et $g \approx g'$, $\min(f, g) \approx \min(f', g')$, etc...

Nous sommes maintenant prêts pour formaliser ce qui est suffisant pour décider de la domination sur une classe de structure.

Fait 2.22. Soit $\mathcal{C}$ une classe de structure close par cylindrification, et soit $\mathcal{F}$ une classe de fonctions de coût sur $\mathcal{C}$ telle que :

1. pour tout $L \subseteq \mathcal{C}$ définissable en logique monadique, χ_L appartient effectivement à $\mathcal{F}$,
2. $\mathcal{F}$ contient la fonction de coût card_X,
3. $\mathcal{F}$ est effectivement close sous cylindrification, minimum, maximum, inf-projection et sup-projection,
4. $\preccurlyeq$ est décidable sur $\mathcal{F}$,

alors les problèmes d'existence de borne, de divergence ainsi que de domination sont décidables pour la logique monadique de coût sur $\mathcal{C}$.

Il s'agit d'une conséquence directe du Fait 2.6. Celui-ci montre comment évaluer une formule de la logique monadique de coût par induction sur la structure des opérations. Chacune des étapes de l'induction correspond à l'une des opérations utilisées ci-dessus.

Chapitre 3

Structure de ce document

Ce document comporte quatre parties. La première s'achève par ce chapitre dont l'objectif est de décrire le reste de ce document.

La partie II traite de la notion de fonction de coût régulière sur les mots.

Chapitre 4 : Les *monoïdes de stabilisation*

Il s'agit dans ce chapitre de résoudre la logique monadique de coût dans le cas des mots par une méthode purement algébrique. L'objet central dans cette résolution est la notion de *monoïde de stabilisation*. Elle nous permet de définir la notion de fonction de coût *reconnue par un monoïde de stabilisation*. Nous résolvons le cas des mots finis et infinis en montrant que toute fonction de coût sur les mots est définie par une formule de la logique monadique de coût est reconnue par un monoïde de stabilisation.

Chapitre 5 : Les *automates à coût*

L'objet de ce chapitre est d'introduire, dans le cas des mots, des notions d'automates correspondant en terme d'expressivité aux monoïdes de stabilisation. Ces automates sont apparus historiquement avant la notion de logique monadique de coût et les monoïdes de stabilisation, mais, à posteriori, ne sont pas nécessaires à sa résolution. Ces objets deviendront centraux dans l'étude de la logique monadique de coût sur les arbres.

Chapitre 6 : *Caractérisations de classes*

Un chapitre riche de la théorie des automates classiques concerne la caractérisation de familles de langages réguliers. Ce chapitre est l'occasion de développer certaines questions de cet ordre dans le cadre des fonctions régulières de coût. Ceci motivera en particulier l'introduction de la notion de *monoïde de stabilisation syntaxique* qui joue dans le cadre des fonctions de coût le rôle du monoïde de stabilisation pour les langages. Nous verrons

alors comment caractériser le *fragment temporel* (fragment dans lequel seuls les intervalles peuvent être mesurés, et non tous les ensembles), ainsi qu'un résultat comparable à celui de Schützenberger, McNaughton et Papert.

La partie III traite des fonctions régulières de coût sur les arbres. Les résultats de cette partie nécessitent d'avoir traité le cas des mots auparavant, et également d'utiliser des techniques radicalement différentes comme les jeux.

Chapitre 7 : Jeux de coût

Dans ce chapitre, nous étudions les notions de jeux de coût. Ce sont des jeux à deux joueurs antagonistes joués sur une arène (un graphe), dans lequel l'objectif des joueurs est mesuré quantitativement à la manière des automates à coût. Tout comme dans le cas de la théorie classique, l'utilisation des jeux est un passage obligé pour comprendre finement le comportement des automates sur les arbres.

Ce chapitre sera également l'occasion d'introduire une forme particulière d'automates à coût : les automates *déterministes en histoire*. Ces automates, bien que formellement non-déterministes, possèdent des propriétés importantes habituellement attribuées aux automates déterministes. En particulier, ils se «composent» bien avec les jeux. Ces notions sont cruciales pour aborder le chapitre suivant.

Chapitre 8 : Fonctions régulières de coût sur les *arbres finis*

Dans ce chapitre, nous résolvons la logique monadique de coût sur les arbres finis. Nous verrons également comment cela permet de résoudre le problème de la hauteur d'étoile et de l'imbrication de point-fixe sur les arbres finis.

Chapitre 9 : Fonctions régulières de coût sur les *arbres infinis*

Ce chapitre montrera la limite des connaissances actuelles sur les fonctions régulières de coût. En effet, le problème de résoudre la logique monadique coût sur les arbres infinis (l'équivalent du résultat tant acclamé de Rabin [87]) est toujours ouvert. Nous expliquerons l'origine de cette difficulté. Nous verrons également comment la résolution de ce problème permettrait de résoudre le problème de la hiérarchie de Mostowski (également ouvert à ce jour).

Chapitre 10 : Logique monadique et automates de coût *faible* et *compteur-faible* sur les mots et les arbres infinis

La logique monadique de coût faible n'autorise comme valuation des variables monadiques que des ensembles finis. Pour cette logique la décidabilité est connue [12]. Nous

verrons d'autres caractérisations de cette classe, ainsi que son extension naturelle par les automates compteurs-faibles.

La partie IV sera l'occasion de conclure.

Chapitre 11 : diverses pistes

En guise de conclusion, nous présentons dans ce chapitre quelques pistes de recherches actives actuellement concernant les fonctions régulières de coût et leurs variantes. Ce sera l'occasion de montrer la richesse des thèmes encore très partiellement soulevés au cours des travaux ayant mené à cette thèse.

Deuxième partie

Les fonctions régulières de coût sur les mots

Chapitre 4

Monoïdes de stabilisation

Nous avons vu au cours du chapitre précédent la logique monadique de coût. Dans ce chapitre nous résolvons le cas des mots finis en décrivant la classe des fonctions de coût reconnaissables. Cette classe de fonctions nous donne, en appliquant l'approche générique décrite dans le fait 2.22, la décidabilité de la domination pour la logique monadique de coût sur les mots. L'objet central de cette définition est algébrique, il s'agit des monoïdes de stabilisation.

Une description alternative des monoïdes de stabilisation a été proposée par Szymon Toruńczyk dans sa thèse [102]. Toruńczyk donne une interprétation à cet objet dans le monoïde profini des monoïdes de stabilisation, mettant en lumière la relation étroite de cet objet avec diverses notions topologiques. En particulier, les fonctions régulières de coût y sont vues comme des langages de mots profinis. Les terminologies étant assez différentes, nous n'aurons que peu l'occasion de présenter les relations entre ces deux formalismes dans la suite.

Une autre approche suivant le même esprit et faisant usage de l'analyse non-standard sera présentée comme piste de recherche.

Nous avons fait le choix de commencer cette présentation par l'approche algébrique, *c.-à-d.* les monoïdes de stabilisation. Il est aussi possible d'aborder, et ce indépendamment, la notion de régularité au travers des automates, en se rendant directement au chapitre suivant. Un tel choix est très pertinent pour un lecteur s'intéressant principalement aux arbres. En effet, dans ce cadre, l'approche algébrique n'est plus d'aucune utilité, et les automates deviennent les objets centraux. Ces deux chapitres sont totalement disjoints dans leurs énoncés (bien que, techniquement, certains résultats sur les automates nécessitent de passer par l'algèbre).

4.1 Aperçu du chapitre

L'objectif de ce chapitre est de présenter une notion de reconnaissabilité adaptée aux fonctions de coût. Il s'agit donc de décrire l'objet ainsi que les outils permettant de travailler avec. Ceci nous permet en particulier d'établir que les fonctions de coût reconnaissables sont telles que :

- la fonction caractéristique de tout langage régulier est reconnaissable (proposition 4.19),
- la fonction de coût «cardinal» sur les mots finis est reconnaissable (example 4.20),
- les fonctions de coût reconnaissables sont effectivement closes sous min, max (corollaire 4.24), sous inf-projection (lemme 4.29), et sup-projection (lemma 4.30),
- la relation $\preccurlyeq$ est décidable sur les fonctions de coût reconnaissables (théoreme 4.27).

En appliquant le fait 2.22, on obtient alors directement le théorème suivant.

Théorème 4.1. *La relation de domination est décidable pour la logique monadique de coût sur les mots finis.*

La suite de ce chapitre se décompose comme suit. Nous introduirons les monoïdes de stabilisation au cours de la section 4.2. La section 4.3 aura pour but de donner une sémantique à cet objet, *c.-à-d.* d'introduire les outils nécessaires pour travailler avec. Enfin, la section 4.4 utilisera les monoïdes de stabilisation pour définir les fonctions de coûts reconnaissables. Nous présenterons la notion de morphisme au cours de la section 4.5. La section 4.6 introduira les $\sharp$-expressions, objet important de la théorie permettant en particulier de donner les premiers résultats de décidabilité. Enfin, la section 4.7 présentera les constructions d'inf-projection et de sup-projection.

4.2 Les monoïdes de stabilisation

Un **semigroupe** $\langle S, \cdot \rangle$ est un ensemble S muni d'une opération de produit associative $(a \cdot (b \cdot c) = (a \cdot b) \cdot c)$. S'il contient un élément neutre 1, *i.e.*, tel que $1 \cdot x = x \cdot 1 = x$ pour tout x, il est appelé **monoïde**. Un **semigroupe ordonné** $\langle S, \cdot, \leq \rangle$ est un semigroupe S équipé d'un ordre $\leq$ compatible, *c.-à-d.* tel que $a \leq b$ et $a' \leq b'$ implique $a \cdot a' \leq b \cdot b'$. Un **idempotent** d'un semigroupe est un élément e tel que $e \cdot e = e$. L'ensemble des idempotents d'un semigroupe $\mathbf{S}$ est noté $E(\mathbf{S})$.

Étant donné un semigroupe $\mathbf{S}$, $\pi_{\mathbf{S}}$ (ou plus simplement π en pratique) est la fonction de $\mathbf{S}^+$ (les mots non-vides sur l'alphabet $\mathbf{S}$) dans $\mathbf{S}$ définie par $\pi_{\mathbf{S}}(a) = a$ et $\pi(u) = \pi_{\mathbf{S}}(u) \cdot \pi_{\mathbf{S}}(a)$ La définition s'étend aux monoïdes en posant $\pi(\varepsilon) = 1$. La fonction π envoie alors tout mot de $\mathbf{M}^*$ (les mots, possiblement vides, sur l'alphabet $\mathbf{M}$) dans $\mathbf{M}$.

Remarque 4.2 (de notation). Il est très important pour la suite de ce chapitre de faire attention aux notations. Nous utilisons dans cette thèse la notation explicite $a \cdot b$ pour dénoter le produit de a avec b. La notation ab est quant-à-elle réservée à la concaténation de lettres. Cela diffère de beaucoup de travaux sur les monoïdes dans lesquels la notion

ab dénote implicitement le produit. Ainsi, pour a, b dans un monoïde $\mathbf{M}$, $a \cdot b$ dénote un élément de M, alors que ab dénote un mot sur l'alphabet M, *c.-à-d.* un élément de $\mathbf{M}^*$. Le produit π permet de passer d'une notation à l'autre.

Définition 4.3. Un **semigroupe de stabilisation** $\langle S, \cdot, \leq, \sharp \rangle$ est un semigroupe ordonné *fini* $\langle S, \cdot, \leq \rangle$ muni d'un opérateur $\sharp$: $E(\mathbf{S}) \to E(\mathbf{S})$ (appelé **stabilisation**) tel que :
- pour tous $e \leq f$ dans $E(\mathbf{S})$, $e^\sharp \leq f^\sharp$;
- pour tous $a, b \in S$ tels que $a \cdot b \in E(\mathbf{S})$ et $b \cdot a \in E(\mathbf{S})$, on a (**consistence**) :

$$(a \cdot b)^\sharp = a \cdot (b \cdot a)^\sharp \cdot b \ ;$$

- pour tout $e \in E(\mathbf{S})$, $e^\sharp \leq e$;
- pour tout $e \in E(\mathbf{S})$, $(e^\sharp)^\sharp = e^\sharp$.

Un **monoïde de stabilisiation** est un semigroupe de stabilisation dont le semigroupe sous-jacent est un monoïde, et tel que $1^\sharp = 1$.

Historiquement, la notion de monoïde de stabilisation est récente [25]. Les idées qui la sous-tendent sont plus anciennes. Leung a le premier introduit la notion de stabilisation afin de résoudre le problème d'existence de borne [63, 65]. Cette technique a ensuite été développée et affinée, en particulier par Simon et Kirsten [98, 52]. L'opérateur de stabilisation dans tous ces travaux est un outil servant à capturer l'aspect quantitatif du problème à étudier. La nouveauté principale des monoïdes de stabilisation est qu'ils sont axiomatisés, et qu'il est possible de leur donner un sens à part entière sans qu'ils ne soient définis, par exemple, à partir d'un automate ou d'une formule logique. Ainsi, ils acquièrent le statut d'objets autonomes dans la théorie.

Pour l'instant, la définition de monoïde de stabilisation est purement abstraite, et il n'est pas évident, *à priori*, de comprendre sa relation avec la logique monadique de coût du chapitre précédent. Nous allons commencer par donner un sens intuitif à cet objet au moyen d'exemples. La section suivante, la section 4.3, aura pour but de formaliser les différentes notions.

Exemple 4.4 («compter» les occurrences de la lettre a)**.** On cherche à construire un monoïde de stabilisation de manière informelle, en ayant pour objectif de décrire la fonction de coût de la fonction calculant le nombre d'occurrences de la lettre a, sur l'alphabet $\{a, b\}$. Informellement, on cherche à construire un monoïde de stabilisation qui sépare les mots contenants «beaucoup» d'occurrences de la lettre a, de ceux qui n'en contiennent que «peu». Cette séparation entre «peu» et «beaucoup» peut sembler manquer de précision et ne dire que très peu sur la fonction que l'on cherche à représenter. Il s'agit en fait exactement de l'information nécessaire à la compréhension des fonctions de coût.

Pour atteindre cet objectif, il faut séparer 3 «types» de mots :
- les mots sans occurrences de la lettre a ; appelons ce type «b» (la lettre b est de ce type).

- les mots contenant au moins une occurrence de la lettre a, mais seulement un petit nombre de telles occurrences ; appelons «a» le type correspondant (en effet, le mot a est de ce type),
- les mots contenant un «grand nombre» d'occurrences de la lettre a's' ; appelons ce type «0».

Notre objectif est de séparer les mots de type 0 des mots de type a ou b. On pose $M = \{a, b, 0\}$, l'ensemble support de notre monoïde.

Bien entendu, concaténer des mots qui ne contiennent aucun a ne peut produire que des mots qui ne contiennent aucun a. Ce phénomène se capture par l'égalité $b \cdot b = b$. Concaténer un mot avec peu de a avec un mot sans a produit un mot avec peu de a. Ainsi, $a \cdot b = b \cdot a = a$. Similairement, concaténer un mot contenant peu de a avec un mot contenant peu de a produit un mot contenant peu de a. Ceci se traduit par l'égalité $a \cdot a = a$. De même, concaténer un mot contenant beaucoup de a avec n'importe quel autre mot produit un mot contenant beaucoup de a. Cela se traduit par $0 \cdot x = x \cdot 0 = 0$ pour tout $x = a, b, 0$. Ainsi, nous avons équipé l'ensemble M d'une opération de produit, et il est aisé de vérifier qu'elle est associative. De plus b en est l'élément neutre. Nous avons donc construit le monoïde $\langle \{a, b, 0\}, \cdot \rangle$.

Il s'agit maintenant d'équiper ce monoïde d'un opérateur de stabilisation. L'opérateur de stabilisation n'est défini que pour les idempotents. Dans le cas présent, tous les éléments sont des idempotents. L'idée principale est la suivante :

$$e^\sharp \quad \text{représente le résultat de l'itération d'un grand nombre de } e \ .$$

Fort de cette interprétation, construire l'opérateut $\sharp$ est très simple. En effet, itérer un grand nombre de fois un mot ne contenant aucune occurrence de la lettre a produit un mot sans occurrence de la lettre a, *i.e.*, $b^\sharp = b$. Itérer un grand nombre de mots contenant un petit nombre de a (au moins 1) produit un mot contenant un grand nombre de a. On pose donc $a^\sharp = 0$. De même itérer un grand nombre de mots contenant un grand nombre de a produit un mot contenant beaucoup d'occurrences de la lettre a. On pose $0^\sharp = 0$.

Les opérateurs de produit et de stabilisation se résument alors dans la table suivante, qui à la paire x, y associe $x \cdot y$, et à chaque x idempotent associe $x^\sharp$:

$x \backslash y$	b	a	0	$\sharp$
b	b	a	0	b
a	a	a	0	0
0	0	0	0	0

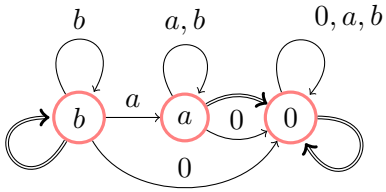
La partie gauche du tableau représente le produit, la colonne supplémentaire donne la table de stabilisation (remarquons que, dans le cas général, il se peut que cette colonne ne soit que partiellement définie, car la stabilisation n'est définie que sur les idempotents).

Pour compléter cette définition, il reste à décrire la relation d'ordre. Par définition d'un monoïde de stabilisation, il est nécessaire de poser $0 = a^\sharp \leq a$. On peut vérifier que cette

relation (augmentée de $x \leq x$ pour tout x) aboutit bien à la définition d'un monoïde de stabilisation.

L'idée qu'il faut avoir de cet ordre est qu'il précise comment la valeur d'un mot peut évoluer quand le curseur séparant «un peu» de «beaucoup» varie. Par exemple, le mot a^{100} doit être considéré comme contenant peu d'occurrences de la lettre a si ce curseur est fixé à 1000, et donc doit avoir la valeur a. En revanche, si l'on fixe le curseur à 10, ce même mot doit être considéré comme ayant beaucoup d'occurrences de la lettre a, et donc doit avoir la valeur 0. Ainsi, il y a une sorte de «continuum» entre la valeur a et la valeur 0, puisqu'un même mot peut passer de l'un à l'autre quand le curseur varie. L'ordre $\leq$ capture cette continuité.

Il est souvent pratique de représenter le monoïde de stabilisation au moyen de son graphe de Cayley :



Chaque sommet représente l'un des éléments du monoïde, et un arc étiqueté y relie chaque sommet x au sommet $x \cdot y$. Une double flèche relie chaque idempotent x à son «stabilisé» $x^\sharp$.

Nous avons déjà souligné, dans le cadre logique, la relation entre la théorie des langages et les fonctions de coût. Cette relation se traduit aussi par le fait que tout monoïde peut être vu comme un monoïde de stabilisation, et il existe une manière canonique de le faire.

Remarque 4.5. Un monoïde au sens usuel $\langle M, \cdot \rangle$ se transforme naturellement en un monoïde de stabilisation $\langle M, \cdot, \sharp, \leq \rangle$, dans lequel la stabilisation de tout idempotent e est $e^\sharp = e$, et l'ordre est réduit à l'identité. On vérifie sans difficulté que cette définition satisfait bien toutes les contraintes d'un monoïde de stabilisation. Nous verrons par la suite que cette traduction préserve également le sens que l'on donne un à monoïde (voir proposition 4.19).

Avant de s'intéresser à la sémantique des monoïdes de stabilisation, donnons un dernier exemple de monoïde de stabilisation.

Exemple 4.6. Nous cherchons cette fois à construire le monoïde de stabilisation correspondant à la fonction de coût qui à chaque mot sur l'alphabet $\{a, b\}$ associe la longueur du plus long segment de lettres a consécutives. Il s'agit donc de séparer les mots dont tous les segments (de a consécutifs) sont petits des mots contenants un grand segment.

Il convient pour effectuer cette séparation de distinguer quatre type de mots :

– le mot vide ; soit «1» l'élément correspondant, (il s'agit de l'élément neutre, noté 1 par convention)

- les petits mots composés uniquement de «a» (d'au moins une lettre) ; soit «a» l'élément correspondant,
- les mots contenant au moins une occurrence de la lettre b, mais dont tous les segments de a's sont petits ; soit «b» l'élément correspondant,
- les mots contenant un grand segment de a consecutifs ; soit «0» l'élément correspondant (il s'agit d'un élément absorbant, noté 0 par convention).

Il n'est pas difficile de vérifier que les opérateurs de produits et de stabilisation correspondants sont ceux donnés par la table suivante :

	1	a	b	0	$\sharp$
1	1	a	b	0	1
a	a	a	b	0	0
b	b	b	b	0	b
0	0	0	0	0	0

Pour compléter la définition du monoïde de stabilisation il reste à donner la relation d'ordre correspondante. Pour cela, il est nécessaire d'avoir $0 \leq a$, car $0 = a^\sharp \leq a$. Comme de plus, $0 \cdot b = b$, on obtient $0 = 0 \cdot b \leq a = b$. L'objet ainsi construit satisfait toutes les propriétés d'un monoïde de stabilisation.

Remarque 4.7. À quoi sert l'ordre $\leq$? Dans les explications informelles que nous avons vues cet ordre semble jouer un rôle secondaire. En effet, il est toujours construit *a posteriori*.

Remarquons tout d'abord que, si l'on dispose d'un monoïde de stabilisation $\mathbf{M} = \langle M, \cdot, \sharp, \leq \rangle$, il se peut qu'il existe un autre ordre $\leq'$ tel que $\langle M, \cdot, \sharp, \leq' \rangle$ soit également un monoïde de stabilisation valide. Appelons un tel ordre **valide**. On remarque sans difficulté que l'intersection de deux ordres valides est aussi un ordre valide. Cela signifie que s'il existe un ordre valide, alors il existe un plus petit ordre valide. Il n'existe par contre pas toujours de plus grand ordre valide.

Cependant, il est important de garantir l'existence d'un ordre valide. En effet, il est possible de construire un ensemble M fini, équipé d'un produit et d'un opérateur de stabilisation tel que tous les axiomes des monoïdes de stabilisation n'impliquant pas l'ordre soient satisfaits, mais pour lesquels il n'existe aucun ordre valide. Dans ce cas, les constructions développées dans la suite de ce chapitre n'ont plus aucun sens. Cela ne signifie pas qu'un tel objet soit sans intérêt, mais plutôt qu'il nécessiterai une étude complètement différente, et qu'il est d'une autre nature. Décrivons un tel exemple.

Imaginons que l'on cherche à construire un monoïde de stabilisation qui caractérise les mots qui «contiennent un nombre pair de petit segments de a consécutifs» (attention, il s'agit d'un exercice de style, car cela ne correspond à aucune fonction). Il est possible de faire entièrement la construction d'un produit et d'une opération de stabilisation comme dans les cas précédents. Ce monoïde doit alors distinguer les types de mots suivants :

- le mot vide, noté «1» (que l'on ne mentionnera plus),
- les mots de a^+ de «petite taille», notés «a»,

– les mots de a^+ de «grande taille», notés «0» (même si cet élément n'est pas absorbant),
– les mots contenant un grand nombre pair (*resp.* impair) de petits blocs de a, et, commençant (*resp.* ne commençant pas)/finissant (*resp.* ne finissant pas) par un grand bloc de a, ce qui correspond aux types b, $0b$, $b0$, $0b0$ et bb, $0bb$, $bb0$, $0bb0$.

La table de multiplication s'obtient en appliquant les égalités suivantes comme des récritures : $a0 = 0a = 00 = 0$, $ba = ab = b$, $bbb = b$ et $b0b = b$. Ainsi, si l'on cherche à calculer $0b0 \cdot 0b0$, on obtient $0b00b0$ qui se récrit en $0b0b0$ puis en $0b0$, en effet, ce type correspond à un mot contenant un nombre pair de petits segments et qui ne commence ni ne finit par un petit segment. Cette opération est associative (cela revient à montrer la confluence du système de récriture). L'opération de stabilisation est alors définie par $a^\sharp = 0^\sharp = 0$, et comme l'identité sur les autres idempotents que sont $bb, 0b, b0$ et $0b0$. La encore, il est possible de montrer que toutes les propriétés des monoïdes de stabilisation sont satisfaites par cette opération de stabilisation.

Le problème surgit quand il s'agit de décrire l'ordre. En effet, il est nécessaire d'avoir d'une part

$$b = b \cdot a^\sharp \cdot b \leq b \cdot a \cdot b = bb \ ,$$

et d'autre part

$$bb = bb \cdot a^\sharp \cdot b \leq bb \cdot a \cdot b = b \ .$$

Ainsi $b \leq bb \leq b$ ce qui contredit l'antisymétrie requise pour faire de $\leq$ un ordre. Il n'existe donc pas d'ordre valide permettant de terminer la construction.

Une manière de se convaincre qu'il y a un problème est de supposer que «être petit» s'interprète comme avoir au plus N éléments. Alors pour chaque valeur de N fixée, il s'agit de représenter l'ensemble

$$L_N = \{a^{n_0}ba^{n_1}b\cdots ba^{n_k} \ : \ \text{il y a un nombre pair de } i = 0, \ldots, k \text{ tels que } n_i \leq N\} \ .$$

Considérons alors un mot de la forme $baba^2ba^3 \ldots ba^n$. Ce mot n'appartient pas à L_0. Il appartient à L_1. Il n'appartient pas à L_2, etc... Ainsi cette propriété n'est pas monotone, et contredit donc le fait 2.3. Ce que cela signifie est qu'il est impossible d'exprimer une telle propriété en logique monadique de coût, et en particulier en respectant la contrainte de positivité qui s'applique aux prédicats de cardinalité. Imposer l'existence de cet ordre revient à éliminer ce type de phénomènes.

4.3 Sémantique des monoïdes de stabilisation

Nous avons vu au cours de la section précédente la définition des monoïdes de stabilisation. Il s'agit maintenant de donner une sémantique à cet objet. Une première solution à ce problème est développée dans le papier original sur le sujet [25]. La solution proposée ici est celle de le version journal [28]. Elle a été développée dans le cadre de l'enseignement

dans le «Master de Recherche Parisien en Informatique». Une autre approche a été proposée par Toruńczyk, utilisant le monoïde profini libre. Elle est combinatoirement rigoureusement équivalente (et utilise les lemmes clefs de [28]), mais possède l'élégance d'une présentation abstraite. Elle sera brièvement abordée au cours de la section 1.5.

L'objet central est la notion de n-calcul. On se fixe pour la suite un monoïde de stabilisation $\mathbf{M} = \langle M, \cdot, \sharp, \leq \rangle$.

Considérons un mot $u \in M^+$ (*c.-à-d.* un mot sur M, vu comme un alphabet). L'objectif est de donner une «valeur» au mot u. Dans le cas des monoïdes, la valeur du mot est naturellement le produit des éléments du mot, *c.-à-d.* $\pi(u)$. Quelle doit être la notion correspondante dans les monoïdes de stabilisation ?

Dans le cas des monoïdes de stabilisation, ce qu'est une valeur nécessite quelques précisions. Toute la sémantique informelle que nous avons vue pour les monoïdes de stabilisation était basée sur la distinction entre «peu» et «beaucoup». Cela signifie que la valeur que prend un mot dépend de ce que l'on considère comme «peu» ou comme «beaucoup». Ainsi, cette valeur est paramétrée par un entier n qui, informellement, représente la valeur limite séparant ce qui est considéré comme «peu» de ce qui est considéré comme «beaucoup». Pour chacune des valeurs prises par n, u est susceptible d'être évalué différemment, *c.-à-d.* d'avoir une valeur différente dans le monoïde de stabilisation.

À titre d'exemple, dans le monoïde de stabilisation comptant le nombre d'occurrences de la lettre a développé dans l'exemple 4.4, un même mot, disons a^{10000}, peut s'évaluer en a (signifiant «un petit nombre de a») si n est «grand» devant 10000, ou s'évaluer en 0 (signifiant «un grand nombre de a») si n est «petit» devant n. Nous verrons que les notions nécessitent néanmoins d'être manipulées avec précaution. En effet, ce qui se passe «autour» de 10000 (pour une notion de «autour» qu'il convient de définir correctement) n'est pas réellement spécifié. Il s'agit d'une zone grise, inévitable, avec laquelle il faut composer. Cette zone grise explique pourquoi les monoïdes de stabilisation ne servent pas à décrire des fonctions, mais des fonctions de coût. L'existence de cette zone grise correspond à la relation d'équivalence $\approx$ dans la définition des fonctions de coût. Revenons donc à la description de cette sémantique.

Supposons qu'une telle valeur limite n soit donnée. La question cruciale reste en suspens : comment détermine-t-on la valeur d'un mot ? La réponse que nous utilisons est de produire un arbre de calcul apportant la preuve que le mot peut s'évaluer en une certaine valeur. Dans le cas des monoïdes usuels, le fait qu'un mot u s'évalue en $\pi(u)$ peut être représenté comme un arbre binaire dont les nœuds sont étiquetés par des valeurs du monoïde, tel que les feuilles lues de gauche à droite forment le mot u, et que chaque nœud binaire soit étiqueté par le produit des étiquettes de ses fils. Bien entendu la racine d'un tel arbre est étiquetée par $\pi(u)$. L'arbre en tant que tel peut être vu comme une «preuve» que u s'évalue en $\pi(u)$. La notion de n-calcul définie ci-dessous est un raffinement de cet notion d'arbre de calcul, adaptée aux monoïdes de stabilisation. Elle est principalement inspirée des arbres de factorisation introduits par Simon [97], en combinaison avec certaines idées issues des travaux de Kirsten [52] (travail dans lequel il n'existe pas de notion équivalente).

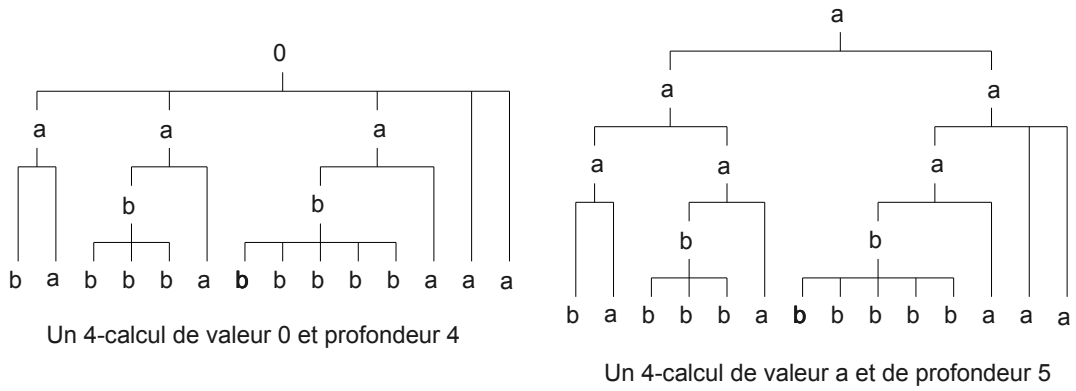


FIGURE 4.1 – Deux arbres de 4-calcul sur le mot *babbbabbbbbaaa* dans le monoïde comptant le nombre d'occurrences de a.

Définition 4.8. Un n**-(arbre de) calcul sur un mot** $u \in M^+$ est un arbre d'arité non fixée, ordonné, dont chaque nœud x est étiqueté par un élément $v(x) \in M$ appelé sa **valeur** et tel que :

- les étiquettes des feuilles lues de gauche à droite forment le mot u,
- pour chaque noeud x, de fils $y_1, \ldots, y_k$ (de gauche à droite), l'une des situations suivantes est vérifiée :
  - $k = 0$ (**feuille**),
  - $k = 2$, et $v(x) = v(y_1) \cdot v(y_2)$, (**nœud produit**),
  - $2 < k \leq n$, et $v(x) = v(y_1) = \cdots = v(y_k) \in E(M)$, (**nœud idempotent**),
  - $k > n$, $v(y_1) = \cdots = v(y_k) = e \in E(M)$ et $v(x) = e^\sharp$ (**nœud de stabilisation**).

La **valeur** d'un arbre de calcul est la valeur de sa racine.

Par convention, on admettra qu'il existe un n-calcul sur le mot vide de valeur 1.

Exemple 4.9. Deux exemples de calcul sont présentés à la figure 4.1. Ils correspondent au monoïde de stabilisation de l'exemple 4.4 dont l'objectif est d'évaluer le nombre d'occurrences de la lettre a dans un mot. Ainsi, étant donné un mot $u \in M^+$ et n, chaque n-calcul décrit une stratégie d'évaluation aboutissant à un résultat. Ce que nous montrent les n-calculs de la figure 4.1, c'est que deux n-calculs (pour la même valeur de n) sur le même mot peuvent avoir des valeurs différentes.

Cet exemple illustre un premier problème lié à l'usage des calculs : il n'y a pas unicité du résultat des n-calculs. Il s'agit de la «zone grise» mentionnée précédemment.

Nous pouvons remarquer aussi que, sur tout mot, il est facile de construire un arbre de calcul : pour cela il suffit de n'utiliser que des nœuds binaires, et jamais de nœuds de stabilisation ou de nœuds idempotents. Bien entendu la valeur d'un tel arbre de calcul n'est

rien d'autre que le produit des lettres du mot. Un tel arbre de calcul ne tient pas compte de l'aspect quantitatif du modèle. Bien sûr, de tels arbres de calcul ne sont pas pertinents. Il convient donc de résoudre les questions suivantes :

- Quels sont les n-calculs pertinents ?
- Est-ce que tout mot admet un n-calcul pertinent pour tout n ?
- Comment relier les différentes valeurs des différents n-calculs pertinents sur un même mot ?

La réponse à la première question est de «limiter» la hauteur des calculs. L'idée est qu'à chaque niveau de l'arbre, «une partie de la précision est perdue». Borner la hauteur des calculs est un moyen de limiter cette perte de précision.

La réponse à la deuxième question est le sujet du théorème 4.10 :

Théorème 4.10 ([28]). *Pour tout mot $u \in M^+$ et tout entier n, il existe un n-calcul de hauteur au plus $3|M|$.*

Nous admettons ce résultat ici. Il se démontre en utilisant la théorie des idéaux dans les monoïdes, *c.-à-d.* les relations de Green. Il s'agit en fait d'une variation autour d'un résultat de Simon sur les monoïdes appelé **théorème des forêts de factorisations** [97]. Le théorème de forêt de factorisation correspond au théorème 4.10 dans le cas d'un monoïde usuel. Dans ce cas il n'y a pas de distinction entre nœuds idempotents ou de stabilisations puisque le stabilisé de tout idempotent est lui-même. Les n-calculs ne dépendent alors plus de n, et sont appelés **arbres de factorisation**. Il existe plusieurs autres preuves cet important résultat. Citons [57, 24, 5, 27] (voir [5] et [27] pour des applications de ce théorème). Il est assez simple de les adapter pour obtenir une preuve du théorème 4.10.

Exemple 4.11. Si nous ne prouvons pas le théorème 4.10 dans ce document, nous pouvons expliquer comment l'obtenir dans le cas de l'example 4.4. Considérons un mot u de $\{a, b, 0\}^*$, et montrons comment construire un n-calcul sur ce mot de hauteur au plus 7. Cela se fait pour des mots de «complexité» croissante.

– Si le mot u appartient à b^+, il existe un calcul de hauteur 1 pour u. En effet, si le mot se réduit à b, il possède bien entendu un calcul de hauteur 0. Si $1 < |u| \leq n$ alors il suffit de construire un n-calcul de hauteur 1 dont la racine est un nœud idempotent de valeur b. Enfin, si $|u| > n$, la même construction convient, le nœud racine ayant toujours la valeur b, mais correspondant cette fois-ci à un nœud de stabilisation.

– Si u appartient à ab^*, alors il possède un calcul de hauteur au plus 2 de valeur a. Si u est simplement a, c'est évident. Sinon, il suffit de construire un nœud binaire de racine a, de fils gauche de valeur a, et dont le fils droit est racine d'un calcul de hauteur au plus 1 pour le mot b^+ comme donné par le cas précèdent.

– Si un mot u est de la forme $(ab^*)^+$, alors il possède un calcul de hauteur au plus 3, et de valeur a ou 0. En effet, ce mot s'écrit $u_1 \dots u_k$, où chaque u_i appartient à ab^*. Chacun des u_is possède un calcul de hauteur au plus 3 d'après le cas précédent. Il suffit alors de construire un arbre dont la racine est a ou 0, possédant k fils étiquetés a, chacun étant

racine d'un sous-calcul pour le mot u_i correspondant. Si $k \leq n$, ce nœud est idempotent, et la valeur de l'arbre est a, sinon, il est de stabilisation, et la valeur du calcul est 0.

– Si un mot u appartient à $(a+b)^+$, alors le seul cas qui reste à traiter correspond à $u \in b^*(ab^*)^+$. En utilisant un nœud binaire à la racine, dont le fils gauche est racine d'un calcul pour le préfixe dans b^+ et le fils droit d'un calcul pour le suffixe dans $(ab^*)^+$. Le calcul correspondant a une hauteur au plus égale à 4.

– Si un mot est de la fome $0(a+b)^*$, en se ramenant au cas précédent, il possède un calcul de hauteur au plus 5, et de valeur 0.

– Si un mot u commence par la lettre 0, alors il s'écrit comme une concaténation de $u_1 \dots u_k$ dans laquelle chacun des u_i appartient à $0(a+b)^*$, et possède donc un calcul de hauteur au plus 5. En construisant un calcul dont la racine est soit un nœud de stabilisation, soit un nœud idempotent, u possède un calcul une hauteur d'au plus 6.

– Le seul cas manquant est de la forme $b^+0(a+b+0)^*$, et se ramène au cas précédent en rajoutant éventuellement un nœud racine binaire. Le résultat est un calcul dont la hauteur est inférieure à 7.

Tous les cas ont été traités.

La réponse à la troisième question est la plus délicate. Pour être énoncée dans toute sa généralité, il convient de raffiner d'abord la notion de n-calcul et d'introduire les n-sous-calculs et les n-sur-calculs. Ces notions correspondent aux calculs, mais combinés à une sur-approximation/sous-approximation du résultat à chaque nœud de l'arbre.

Définition 4.12. Un **n-sous-calcul sur un mot** u est un arbre d'arité non fixée, ordonné, dont chaque nœud x est étiqueté par un élément $v(x) \in M$ appelé sa **valeur** et tel que :

- les étiquettes des feuilles lues de gauche à droite forment un mot $\leq u$ (l'ordre sur M est étendu aux mots de M^* lettre à lettre),
- pour chaque noeud x, de fils $y_1, \dots, y_k$ (de gauche à droite), l'une de ces situations est vraie :
  - $k = 0$ (feuille),
  - $k = 2$, et $v(x) \leq v(y_1) \cdot v(y_2)$ (nœud binaire),
  - $k = 2 \dots n$, et $v(x) \leq v(y_1) = \dots = v(y_k) = v(x) \in E(M)$ (nœud idempotent),
  - $k = n+1 \dots$, $v(y_1) = \dots = v(y_k) = e \in E(M)$ et $v(x) \leq e^\sharp$ (nœud de stabilisation).

Un **n-sur-calcul** est défini identiquement en remplaçant $\leq$ par $\geq$ partout.

La **valeur** d'un arbre de sous-calcul (*resp.* sur-calcul) est la valeur de sa racine.

Par convention, on admettra qu'il existe un n-sous-calcul (*resp.* un n-sur-calcul) sur le mot vide pour toute valeur inférieure ou égale à 1 (*resp.* supérieure ou égale).

L'idée est que les sous-calculs permettent de sous-approximer la valeur d'un mot, les sur-calculs permettent de les sur-approximer. Bien entendu, les notions de sur-calculs et de sous-calculs sont intimement reliées à la notion de calcul comme le montre le fait suivant.

Fait 4.13. Tout n-calcul est un n-sous-calcul et un n-sur-calcul, et réciproquement, toute arbre étiqueté qui est à la fois un n-sous-calcul et un n-sur-calcul est un n-calcul.

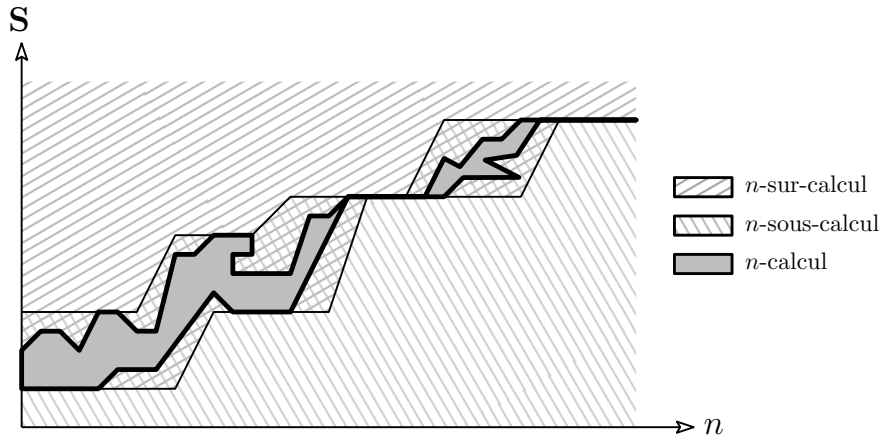


FIGURE 4.2 – Structure des calculs, sous-calculs et sur-calculs.

Les sous-calculs et les sur-calculs ont des propriétés que les calculs n'ont pas, et qui les rendent plus faciles à manipuler.

Fait 4.14. Soit $m \leq n$, alors tout m-sous-calcul est n-sous-calcul, et tout n-sur-calcul est un m-sur-calcul.

Cela s'interprète sur la figure 4.2. Un mot est fixé ainsi qu'une hauteur p supérieure à $3|M|$. On cherche à représenter pour quelles valeurs de n et de a il existe un n-⌊sur/sous-⌋calcul de hauteur au plus p. Les abscisses correspondent à la valeur de n, et les ordonnées à l'ordre (représenté linéairement ici) du monoïde de stabilisation. Les n-calculs de valeur a de hauteur au plus $3|M|$, sont symbolisés par l'appartenance du point d'abscisse n et d'ordonnée a à la zone grise. Les sous-calculs de hauteur au plus $3|M|$, qui sont clos vers le bas et la droite d'après le fait 4.13, délimitent la zone hachurée descendante alors que les sur-calculs de hauteur au plus $3|M|$, sont eux clos vers la gauche et le haut, et correspondent à la zone hachurée de manière ascendante. Le théorème 4.15 suivant énonce que les frontières de ces zones, en gras sur le dessin, ne peuvent être arbitrairement éloignées ; elles se comportent «globalement, de manière identique».

Théorème 4.15 (C. [28]). *Pour tout entier p, il existe un polynôme $\alpha : \mathbb{N} \to \mathbb{N}$ tel que pour tout $\alpha(n)$-sur-calcul de valeur b sur un mot u et tout n-sous-calcul sur le même mot de valeur a de hauteur au plus p, alors*

$$a \leq b \ .$$

Ce théorème se démontre par induction sur la hauteur du sous-calcul. Nous ne développons pas cette preuve dans ce travail.

Ainsi, il faut que la hauteur soit suffisamment grande pour qu'il existe un calcul par le théorème 4.10, mais bornée pour qu'un unique α permette de borner la marge d'erreur par le théorème 4.15.

Exemple 4.16. À titre d'exemple, établissons le théorème 4.15 dans le cas de l'exemple 4.4.

Considérons tout d'abord un n-sur-calcul de valeur v sur un mot u et montrons par induction sur la hauteur que :

($\star_1$) Si $u \in b^+$ alors $v = b$: en listant tous les cas, car $b \cdot b = b^\sharp = b$, et que le seul x tel que $x \geq b$ est $x = b$.

($\star_2$) Si u ne contient aucun 0 et $|u|_a \leq n$, alors $v \in \{a, b\}$: en listant tous les cas. La seule façon de produire l'élément 0 étant par $0 = a^\sharp$, le seul cas problématique est le cas où la racine est un nœud de stabilisation dont tous les fils ont pour valeur a. Mais d'après le cas précédent, cela veut dire qu'une occurrence de a apparait dans chacun des sur-calculs enracinés au fils de la racine. Comme le nœud racine est de stabilisation, il a strictement plus de n fils, ce qui contredit l'hypothèse $|u|_a \leq n$.

Considérons maintenant un n-sous-calcul de hauteur au plus p de valeur v sur un mot u de $(b + a + 0)^+$. Nous montrons par récurrence :

($\star\star_1$) Si u contient un a, alors $v \leq a$: en listant tous les cas, toutes les opérations impliquant une lettre $\leq a$ ayant un résultat $\leq a$.

($\star\star_2$) Si u contient au moins $n^p + 1$ occurrences de a, alors $v = 0$. Il s'agit de considérer deux cas pour établir ce fait. Si le nœud racine est un nœud de stabilisation. Soit y la valeur d'un/de tous ses fils. D'après le cas précédent, y est nécessairement $\leq a$. Il s'ensuit que $v \leq a^\sharp = 0$. Dans tous les autres cas, comme la racine possède au plus n fils, l'un au moins correspond à un n-sur-calcul de hauteur au plus $p-1$ sur un mot contenant au moins $n^{p-1} + 1$ occurrences de la lettre a. Par récurrence, cela signifie que l'un des fils de la racine a la valeur 0. En inspectant tous les cas, il s'ensuit que la racine a aussi la valeur 0 (0 est absorbant).

($\star\star_3$) Si u contient un 0, alors $v = 0$ (en listant tous les cas).

Soit donc un n^p-sur-calcul de valeur y sur un mot u et un n-sous-calcul sur le même mot de valeur x de hauteur au plus p, il s'agit de montrer que $x \leq y$ (cf. théorème 4.15).

- Si $x = b$, alors d'après $\star\star_1$ et $\star\star_3$ que u n'est constitué que de lettres b. Par $\star_1$, il s'ensuit que $y = b$. Nous avons bien $x \leq y$.
- Si $x = a$, alors, d'après $\star\star_2$ et $\star\star_3$, u ne possède pas d'occurrences de 0, et possède au plus n^p occurrences de a. Il s'ensuit par $\star_2$ que $y \geq a$. De nouveau $x \leq y$.
- Si $x = 0$, alors, $x \leq 0$.

Exemple 4.17. La remarque 4.5 a été l'occasion de présenter comment un monoïde usuel $\langle M, \cdot \rangle$ peut être transformé en un monoïde de stabilisation $\langle M, \cdot, id, = \rangle$. Qu'advient-il des calculs dans ce cas ? Nous obtenons alors que, pour tout n, un n-calcul (*resp.* sur/sous-calcul) sur un mot u a toujours pour valeur $\pi(u)$. Le théorème 4.15 est donc trivialement vérifié dans ce cas puisque tous les sur-calculs et tous les sous-calculs sur un même mot u ont la même valeur, $\pi(u)$.

4.4 Reconnaissabilité

Nous avons vu le modèle des monoïdes de stabilisation, ainsi que les arbres de calculs permettant de capturer leur sémantique. Dans cette section, nous utilisons ces notions pour définir ce qu'est une fonction de coût reconnaissable.

Fixons nous un monoïde de stabilisation $\mathbf{M}$. Un **idéal** de $\mathbf{M}$ est une partie I de M close vers le bas, *c.-à-d.* telle que si $a \leq b$ et $b \in I$ alors $a \in I$. Étant donnée une partie $X \subseteq M$, on note par $X{\downarrow}$ le plus petit idéal qui la contient, *c.-à-d.* $X{\downarrow} = \{y \ : \ y \leq x, \ x \in X\}$.

Les trois ingrédients servant à reconnaître une fonction de coût sont un monoïde de stabilisation $\mathbf{M}$, une application h de l'alphabet $\mathbb{A}$ dans M que l'on étend lettre à lettre en un morphisme $\tilde{h}$ de $\mathbb{A}^*$ dans M^*, et un idéal $I \subseteq M$. On définit alors pour chaque entier p quatre fonctions de $\mathbb{A}^*$ dans $\mathbb{N} \cup \{\infty\}$:

$$\begin{aligned}
[\![\mathbf{M}, h, I]\!]_p^{--}(u) &= \inf\{n \ : \text{ il existe un } n\text{-sous-calcul de valeur dans } M \setminus I \\
&\qquad \text{de hauteur au plus } p \text{ sur } \tilde{h}(u)\} \\
[\![\mathbf{M}, h, I]\!]_p^{-}(u) &= \inf\{n \ : \text{ il existe un } n\text{-calcul de valeur dans } M \setminus I \text{ de} \\
&\qquad \text{hauteur au plus } p \text{ sur } \tilde{h}(u)\} \\
[\![\mathbf{M}, h, I]\!]_p^{+}(u) &= \sup\{n+1 \ : \text{ il existe un } n\text{-calcul de valeur dans } I \text{ sur } \tilde{h}(u)\} \\
[\![\mathbf{M}, h, I]\!]_p^{++}(u) &= \sup\{n+1 \ : \text{ il existe un } n\text{-sur-calcul de valeur dans } I \text{ de} \\
&\qquad \text{hauteur au plus } p \text{ sur } \tilde{h}(u)\}
\end{aligned}$$

Le lemme suivant montre que toutes ses fonctions, pour p suffisamment grand sont équivalentes :

Lemme 4.18. *Pour tout $p \geq 3|M|$,*

$$[\![\mathbf{M}, h, I]\!]_p^{--} \leq [\![\mathbf{M}, h, I]\!]_p^{-} \leq [\![\mathbf{M}, h, I]\!]_p^{+} \leq [\![\mathbf{M}, h, I]\!]_p^{++} \ .$$

Pour tout p, il existe un polynôme α tel que

$$[\![\mathbf{M}, h, I]\!]_p^{++} \leq \alpha \circ [\![\mathbf{M}, h, I]\!]_p^{--} \ .$$

Pour tout $p \leq r$,

$$[\![\mathbf{M}, h, I]\!]_r^{--} \leq [\![\mathbf{M}, h, I]\!]_p^{--} \ , \qquad \text{et} \quad [\![\mathbf{M}, h, I]\!]_p^{++} \leq [\![\mathbf{M}, h, I]\!]_r^{++} \ .$$

Démonstration. Premier item, inégalité centrale. Soit u un mot et m un entier naturel. Supposons que $[\![\mathbf{M}, h, I]\!]_p^{-}(u) > m$, cela signifie qu'il n'existe pas de n-calcul de hauteur au plus p sur $\tilde{h}(u)$ à valeur dans $M \setminus I$ pour $n \leq m$. Pourtant il existe un m-calcul sur $\tilde{h}(u)$ de hauteur au plus p d'après le théorème 4.10. La valeur de ce calcul appartient nécessairement à I. Ce calcul témoigne de l'inégalité $[\![\mathbf{M}, h, I]\!]_p^{+}(u) > m$. Les deux autres inégalités sont

des conséquences directes du fait 4.13 : tout n-calcul est à la fois un n-sous-calcul et un n-sur-calcul.

Le second item est une reformulation du théorème 4.15.

Le troisième item correspond au fait que plus p est grand, plus il y'a de sous-calculs (*resp.*, sur-calculs) de hauteur au plus p. Ainsi, l'infimum définissant $[\![\mathbf{M}, h, I]\!]_p^-$ (*resp.* le supremum définissant $[\![\mathbf{M}, h, I]\!]_p^-$) ne peut que décroître (*resp.* augmenter). CQFD

Plus simplement, on obtient donc que pour tous $p, r \geq 3|M|$,

$$[\![\mathbf{M}, h, I]\!]_p^{--} \approx [\![\mathbf{M}, h, I]\!]_p^- \approx [\![\mathbf{M}, h, I]\!]_p^+ \approx [\![\mathbf{M}, h, I]\!]_p^{++} \approx [\![\mathbf{M}, h, I]\!]_r^{--} \ .$$

Ainsi, toutes ces fonctions appartiennent à la même classe d'équivalence, notée simplement $[\![\mathbf{M}, h, I]\!]$ et appelée la **fonction de coût reconnue par** $\mathbf{M}, h, I$. On dira parfois **reconnue par** $\mathbf{M}$. On dira aussi que $[\![\mathbf{M}, h, I]\!]$ et plus généralement toute fonction dans $[\![\mathbf{M}, h, I]\!]$ est **reconnaissable**.

Ce que l'on voit dans cette définition est qu'un monoïde de stabilisation ne définit pas une fonction en particulier, mais de nombreuses fonctions, qui sont toutes équivalentes. En ce sens, ce modèle est intimement relié à la notion de fonction de coût.

Proposition 4.19. *La fonction de coût caractéristique d'un langage régulier est reconnaissable.*

Démonstration. Considérons un langage L régulier. Qu'il soit reconnaissable signifie qu'il existe un monoïde $\mathbf{M}$, une application $h : \mathbb{A} \to M$ et une partie $F \subseteq M$ tels que :

$$L = \{u \ : \ \pi(\tilde{h}(u)) \in F\} \ ,$$

où $\tilde{h}(u)$ représente l'extension lettre à lettre de h en une application de $\mathbb{A}^*$ dans M^*, et π dénote le produit généralisé dans M.

En suivant l'exemple 4.17, ce monoïde peut être vu comme un monoïde de stabilisation. Choisissons $I = M \setminus F$ comme idéal. Considérons alors la fonction reconnue par $\mathbf{M}, h, I$. Comme tout sous/sur-calcul sur un mot v s'évalue en $\pi(v)$, toutes les fonctions $[\![\mathbf{M}, h, I]\!]_p^{--}$, $[\![\mathbf{M}, h, I]\!]_p^-$, $[\![\mathbf{M}, h, I]\!]_p^+$ et $[\![\mathbf{M}, h, I]\!]_p^{++}$ pour $p \geq 3|M|$ associent à un mot u la valeur 0 si $\pi(\tilde{h}(u)) \in F$ et ∞ sinon. Il s'agit de la fonction de coût caractéristique de L. CQFD

Ce qu'il faut remarquer de cette preuve, c'est que le même monoïde permet de reconnaitre un langage et, quand il est vu comme un monoïde de stabilisation, permet de reconnaitre la fonction caractéristique du langage. Cette relation se poursuit dans la suite de ce chapitre, ainsi, chacune des opérations effectuées sur les monoïdes de stabilisation, quand elle est interprétée dans le cas des monoïdes, correspond à une opérations classiques.

L'exemple suivant utilise les capacités nouvelles des monoïdes de stabilisations comparées aux monoïdes classiques.

Exemple 4.20. Considérons la fonction de coût reconnue par $\mathbf{M}, \{a \mapsto a, b \mapsto b\}, I = \{0\}$, ou $\mathbf{M}$ est le monoïde de stabilisation de l'exemple 4.4. En reprenant les résultats développés au cours de l'exemple 4.16, on obtient que pour tout mot $u \in \{a, b\}^+$,

– si $|u|_a \leq n$, alors tous les n-sur-calculs sur u ont une valeur dans $M \setminus I$, et ;

– si $|u|_a > n^p$, alors tous les n-sous-calculs de hauteur au plus p ont une valeur dans I.

On en déduit que la fonction de coût reconnue ainsi est celle de l'application $|\ \ |_a$, qui à chaque mot associe le nombre d'occurrences de la lettre a.

4.5 Quotients, sous-monoïdes des stabilisation et produits

Il s'agit dans cette section d'introduire certaines définitions algébriques classiques, spécialisées au cadre des monoïdes de stabilisation. Soient $\mathbf{M} = \langle M, \cdot, \sharp, \leq \rangle$ et $\mathbf{M}' = \langle M', \cdot', \sharp', \leq' \rangle$. Un **morphisme de monoïde de stabilisation** de $\mathbf{M}$ dans $\mathbf{M}'$ est une application μ de M dans M' telle que

– $\mu(1_{\mathbf{M}}) = 1_{\mathbf{M}'}$,

– $\mu(x) \cdot' \mu(y) = \mu(x \cdot y)$ pour tous x, y dans M,

– pour tous x, y dans M, si $x \leq y$ alors $\mu(x) \leq \mu(y)$,

– $\mu(e)^\sharp = \mu(e^\sharp)$ pour tout $e \in E(\mathbf{M})$, (remarquons que $\mu(e) \in E(\mathbf{M}')$).

Il est aisé de vérifier que :

Proposition 4.21. *Les n-calculs (resp., n-sous-calculs, n-sur-calculs) sur $\mathbf{M}$ sont transformés par morphisme (le morphisme appliqué à chaque nœud de l'arbre) en n-calculs (resp., n-sous-calculs, n-sur-calculs) sur $\mathbf{M}'$.*

D'où l'on déduit :

Corollaire 4.22. *Si μ est un morphisme de monoïdes de stabilisation de $\mathbf{M}$ dans $\mathbf{M}'$, h est une application d'un alphabet $\mathbb{A}$ dans $\mathbf{M}$ et I' est un idéal de $\mathbf{M}'$, alors*

$$[\![\mathbf{M}, h, \mu^{-1}(I')]\!] = [\![\mathbf{M}', \mu \circ h, I']\!] \ .$$

Démonstration. Remarquons d'abord que $\mu^{-1}(I')$ est bien un idéal de $\mathbf{M}$, et par conséquence l'énoncé a bien un sens.

Soit u un mot de $\mathbb{A}^*$. Considérons un n-calcul sur $\mathbf{M}$ du mot $\tilde{h}(u)$ de valeur $a \in \mu^{-1}(I')$, il peut être transformé par morphisme en un n-calcul sur $\mathbf{M}'$ sur le mot $(\mu \circ h)(u)$ de valeur $\mu(a) \in I'$. De même un n-cacul du mot $\tilde{h}(u)$ de valeur $a \in M' \setminus \mu^{-1}(I')$ peut être transformé en un n-calcul de valeur $\mu(a) \in M' \setminus I'$. CQFD

La notion de morphisme est intimement reliée à la notion de produit. C'est le cas aussi pour les monoïdes de stabilisation. Étant donnés $\mathbf{M} = \langle M, \cdot, \sharp, \leq \rangle$ et $\mathbf{M}' = \langle M', \cdot', \sharp', \leq' \rangle$ deux monoïdes de stabilisation, on définit leur produit $\mathbf{M} \times \mathbf{M}'$ comme :

$$\mathbf{M} \times \mathbf{M}' = \langle M \times M', \cdot'', \sharp'', \leq'' \rangle$$

où $(x,x')\cdot''(y,y') = (x\cdot y, x'\cdot' y')$, $(e,e')^{\sharp''} = (e^\sharp, e'^{\sharp'})$, et $(x,x') \leq'' (y,y')$ ssi $x \leq y$ et $x' \leq' y'$.

Assez naturellement, l'application de projection sur la première (*resp.*, seconde) composante est un morphisme surjectif de $\mathbf{M} \times \mathbf{M}'$ sur $\mathbf{M}$ (*resp.*, sur $\mathbf{M}'$). Il s'ensuit par le corollaire 4.22 que si f est reconnue par $\mathbf{M}, h, I$ et g par $\mathbf{M}', h', I'$, alors f est aussi reconnue par $\mathbf{M} \times \mathbf{M}', h \times h', I \times M'$ et g par $\mathbf{M} \times \mathbf{M}', h \times h', M \times I'$, où l'on pose $(h \times h')(a) = (h(a), h(a'))$ pour toute lettre a. Ainsi, on obtient :

Lemme 4.23. *Si f et g sont reconnues par $\mathbf{M}, h, I$ et $\mathbf{M}', h', I'$ respectivement, alors il existe un monoïde de stabilisation $\mathbf{M}''$ et une application h'' tels que :*
- *il existe un morphisme surjectif f de $\mathbf{M}''$ sur $\mathbf{M}$ tel que $h = f \circ h''$, et*
- *il existe un morphisme surjectif f' de $\mathbf{M}''$ sur $\mathbf{M}'$ tel que $h' = f' \circ h''$.*

Ce qui signifie, en particulier que :
- $[\![\mathbf{M}'', h'', f^{-1}(I)]\!] = [\![\mathbf{M}, h, I]\!]$, *et*
- $[\![\mathbf{M}'', h'', f'^{-1}(I')]\!] = [\![\mathbf{M}', h', I']\!]$.

Une conséquence directe de cette construction est alors :

Corollaire 4.24. *Si f et g sont reconnaissables, alors* $\max(f,g)$ *et* $\min(f,g)$ *sont effectivement reconnaissables.*

Démonstration. D'après le lemma 4.23, on peut supposer f reconnue par $\mathbf{M}, h, I$ et g par $\mathbf{M}, h, J$. Alors (pour la hauteur $p = 3|M|$ fixée) on a :

$$\begin{aligned}
&\max([\![\mathbf{M}, h, I]\!]_p^+, [\![\mathbf{M}, h, J]\!]_p^+)(u) \\
&= \max \begin{cases} \sup\{n \ : \text{ il existe un } n\text{-calcul sur } h(u) \text{ de valeur dans } I\} \\ \sup\{n \ : \text{ il existe un } n\text{-calcul sur } h(u) \text{ de valeur dans } J\} \end{cases} \\
&= \sup\{n \ : \text{ il existe un } n\text{-calcul sur } h(u) \text{ de valeur dans } I \cup J\} \\
&= [\![\mathbf{M}, h, I \cup J]\!]_p^+ \ .
\end{aligned}$$

Ainsi $\max(f,g)$ est reconnue par $\mathbf{M}, h, I \cup J$. De manière similaire, $\min(f,g)$ est reconnue par $\mathbf{M}, h, I \cap J$. CQFD

4.6 Les ♯-expressions

Nous avons vu la notion de monoïde de stabilisation, et comment elle pouvait être utilisée pour définir une notion de fonction de coût reconnaissable. Nous manquons encore de quelques outils pour effectuer des preuves. Le principal outil est la notion de ♯-expression, introduit par Hashiguchi au cours de son étude des automates de distance [42]. Les ♯-expressions peuvent être vues de deux manières. D'une part, comme une expression dénotant un élément dans un monoïde de stabilisation, et d'autre part comme un générateur de séquences infinies de mots pouvant servir comme témoins d'absence d'une borne, de non-divergence, ou plus généralement de non-domination.

Dans un monoïde fini $\mathbf{M}$, étant donné un élément $a \in M$, nous dénotons par a^ω l'unique [1] idempotent puissance de a. En particulier, $a^\omega = a^{|M|!}$. Cette notion est pratique car l'opérateur de stabilisation n'est défini que sur les idempotents. Dans un monoïde de stabilisation, nous noterons $a^{\omega\sharp}$ l'élément $(a^\omega)^\sharp$. Contrairement à $a^\sharp$ qui n'est pas toujours défini, $a^{\omega\sharp}$ l'est toujours [2].

Une **♯-expression** [46] sur un ensemble A est une expression composée de lettres dans A, de produits, et d'exposant par $\omega\sharp$ (originellement, les ♯-expressions utilisent des exposants $\sharp$; l'utilisation de l'exposant $\omega\sharp$ évite certaines complications).

Une ♯-expression E sur un monoïde de stabilisation $\mathbf{M}$ dénote un calcul dans le monoïde de stabilisation. Le résultat $\mathit{valeur}(E)$ de ce calcul est appelé **valeur** de la ♯-expression. Une ♯-expression est **stricte** si elle contient au moins une occurrence de $\omega\sharp$. Étant donné un ensemble $A \subseteq M$, $\langle A\rangle^\sharp$ dénote l'ensemble des valeurs des expressions sur A. De manière équivalente, il s'agit du plus petit ensemble contenant A et clos par produit et stabilisation, *c.-à-d.* le sous-semigroupe de stabilisation engendré par A. On notera également $\langle A\rangle^{\sharp+}$ l'ensemble des valeurs des expressions strictes sur A.

Le n**-dépliage** d'une ♯-expression sur A est le mot de A^+ défini par récurrence comme suit :

$$\begin{aligned}
\text{dépliage}(a,k,n) &= a\\
\text{dépliage}(EF,k,n) &= \text{dépliage}(E,k,n)\text{dépliage}(F,k,n)\\
\text{dépliage}(E^{\omega\sharp},k,n) &= \overbrace{\text{dépliage}(E,k,n)\ldots\text{dépliage}(E,k,n)}^{kn \text{ fois}}
\end{aligned}$$

Il peut sembler étrange d'utiliser deux entiers k et n. En fait il faut interpréter a^ω comme l'itération k fois de a, et $e^\sharp$ comme l'itération n fois de e, pour e idempotent. Dans ce cas, il est naturel d'avoir $a^{\omega\sharp}$ qui corresponde à l'itération kn fois de a.

Étant donnée une ♯-expression E, $\text{Dépliages}^+(E,k)$ est le langage obtenu en évaluant l'expression rationnelle obtenue de E en remplaçant ω par k et $\sharp$ par $+$. Autrement dit,

$$\begin{aligned}
\text{Dépliages}^+(a,k) &= \{a\}\ ,\\
\text{Dépliages}^+(EF,k) &= \text{Dépliages}^+(E,k)\text{Dépliages}^+(E,k)\ ,\\
\text{Dépliages}^+(E^{\omega\sharp},k) &= ((\text{Dépliages}^+(E,k))^k)^+\ .
\end{aligned}$$

Nous concluons cette section en montrant comment les ♯-expressions peuvent être utilisées comme témoins d'existence ou de non-existence d'une borne.

Proposition 4.25. *Supposons que* $\mathbf{M}, h, I$ *reconnaisse une fonction de coût sur l'alphabet* $\mathbb{A}$*, que* k *est un multiple de* $|M|!$*, et soit* E *une* ♯*-expression sur* $\mathbb{A}$*, alors les énoncés suivants sont équivalents :*

1. Cet élément n'existe pas toujours dans le cas des monoïdes infinis. Il existe toujours dans le cas des monoïdes finis, et est toujours unique.
2. Certains auteurs dénotent $\omega\sharp$ simplement $\sharp$.

1. $[\![\mathbf{M}, h, I]\!]$ *est bornée sur Dépliages*$^+(E, k)$,
2. $[\![\mathbf{M}, h, I]\!]$ *est bornée sur* $\{$*dépliage*(E, k, n) : $n \geq 1\}$, *et*
3. *valeur*$(h(E)) \notin I$,

Démonstration. Comme $\{\text{dépliage}(E, K, n) : n \geq 1\} \subseteq \text{Dépliages}^+(E, k)$, l'implication de 1 à 2 est directe.

De 2 à 3. Par la contraposée, supposons que *valeur*$(h(E)) \in I$. Il s'agit, étant donné un entier n, de produire un calcul sur le mot dépliage(E, k, n) témoignant du fait que la valeur de $[\![\mathbf{M}, h, I]\!]_p^+(\text{dépliage}(E, k, n))$ est supérieure à n (pour un p ne dépendant que de E). C calcul, appelé k, n-calcul de E est défini par récurrence comme l'arbre de calcul suivant (on utilise la notation $[a](t_1, \ldots, t_i)$ pour dénoter l'arbre de racine étiquetée par a et dont les fils sont $t_1, \ldots, t_i$, de gauche à droite, et on s'autorise à utiliser temporairement la notation E^ω) :

$$\begin{aligned}
\text{calcul}(a, k, n) &= [a] \ , \\
\text{calcul}(EF, k, n) &= [\mathit{valeur}(EF)](\text{calcul}(E, k, n), \text{calcul}(F, k, n)) \ , \\
\text{calcul}(E^{\omega\sharp}, k, n) &= [\mathit{valeur}(E^{\omega\sharp})](\overbrace{\text{calcul}(E^\omega, \ldots, E^\omega, k, n)}^{n \text{ fils}}) \ , \\
\text{calcul}(E^\omega, k, n) &= \text{calcul}(\overbrace{E \ldots E}^{k \text{ fois}}, k, n) \ .
\end{aligned}$$

On vérifie par une analyse de cas que calcul(E, k, n) est un n-calcul sur le mot dépliage(u, k, n), de valeur *valeur*(E). En particulier, le fait que k soit un multiple de $|M|!$ assure que $a^k = a^\omega$ pour tout a. Dans le cas présent, cela garantit que la valeur de calcul(E^ω, k, n) soit un idempotent, ce qui est nécessaire pour que la définition de calcul$(E^{\omega\sharp}, k, n)$ produise un calcul valide. En fait, ce calcul est un peu plus général : calcul(E, k, n) est un m-calcul sur le mot dépliage(E, k, n) pour tout $m \leq n$. De plus, le hauteur du calcul calcul(E, k, n) est au plus $k|E|$. Supposons maintenant $a = \mathit{valeur}(h(E)) \in I$. Il s'ensuit que (en omettant les hauteurs des calculs, toutes bornées par $k|E|$) :

$$\begin{aligned}
[\![\mathbf{M}, h, I]\!]^+_{k|E|} &= \sup\{m + 1 \ : \ \text{il existe un } m\text{-calcul sur } \tilde{h}(\text{dépliage}(E, k, n)) \\
&\qquad\qquad \text{et de valeur dans } I\} \\
&\geq n \ . \qquad\qquad \text{(en utilisant calcul}(E, k, n)\text{-comme calcul témoin)}
\end{aligned}$$

De 3 à 1. Par récurrence sur la hauteur d'une ♯-expression E, montrons qu'il existe un entier n tel que pour tout mot $u \in \text{Dépliages}^+(E, k)$ il existe un n-sous-calcul sur $\tilde{h}(u)$ de hauteur au plus $(k + 1)|E|$, et de valeur *valeur*(E). Le seul cas intéressant est $E = F^{\omega\sharp}$. Un mot de $u \in \text{Dépliages}(F^{\omega\sharp})$ s'écrit alors comme $u = u_1 \ldots u_\ell$, où $u_1, \ldots, u_\ell \in \text{Dépliages}^+(F, k)$. D'après l'hypothèse de récurrence, sur chacun de ces mots il existe un n-sous-calcul t_i de hauteur au plus $(k+1)(|E|-1)$ et valeur *valeur*(F) sur $\tilde{h}(u_i)$. Construisons

alors, pour $s = 1 \dots k$, le terme t_i^s défini par $t_i^1 = t_i$ et $t_i^{s+1} = [valeur(F)^{i+1}](t_i, t_i^s)$. Le terme t_i^s est un n-sous-calcul sur $h(u_i^s)$ de valeur $valeur(F)^s$. En particulier, pour $k = s$, $valeur(F)^k$ est un idempotent. Il suffit alors de construire le terme

$$t = [valeur(F)^{\omega\sharp}](t_1^k, \dots, t_\ell^k) \ .$$

Il s'agit d'un n-sous-calcul de hauteur au plus $(k+1)|E|$ sur le mot $\tilde{h}(u)$, et de valeur $valeur(E)$. Il s'ensuit directement que pour tout mot $u \in \text{Dépliages}^+(E,k)$, $[\![\mathbf{M}, h, I]\!]^{--}_{3|M|}(u) \leq (k+1)|E|$.

CQFD

De cet énoncé, on obtient une caractérisation simple de la domination entre fonctions reconnues par le même monoïde de stabilisation.

Lemme 4.26. *Soit* $\mathbf{M}$ *un monoïde de stabilisation,* h *une application d'un alphabet* $\mathbb{A}$ *dans* M*, et deux idéaux* I, J*, alors les énoncés suivants sont équivalents :*

- $[\![\mathbf{M}, h, I]\!] \preccurlyeq [\![\mathbf{M}, h, J]\!]$*, et*
- $\langle h(\mathbb{A})\rangle^\sharp \cap I \subseteq J$*.*

Démonstration. De haut en bas. Supposons que $[\![\mathbf{M}, h, I]\!] \preccurlyeq [\![\mathbf{M}, h, J]\!]$, et considérons k un multiple de $|M|!$. Étant donnée une $\sharp$-expression E sur $\mathbb{A}$, notons $h(E)$ l'expression sur M obtenue en remplaçant chaque lettre $b \in \mathbb{A}$ par $h(b)$. Soit $a \in \langle h(\mathbb{A})\rangle^\sharp \cap I$. Alors il existe une $\sharp$-expression E sur $\mathbb{A}$ telle que $valeur(h(E)) = a$. D'après la proposition 4.25, cela signifie que $[\![\mathbf{M}, h, I]\!]$ n'est pas bornée sur Dépliages(E,k). Donc, par hypothèse de domination, $[\![\mathbf{M}, h, J]\!]$ n'est pas non plus bornée sur cet ensemble. De nouveau d'après la proposition 4.25, on en déduit que $a = valeur(h(E)) \in J$.

De bas en haut. Supposons que $\langle h(\mathbb{A})\rangle^\sharp \cap I \subseteq J$. Considérons un mot $u \in \mathbb{A}^*$. Tout calcul sur $\tilde{h}(u)$ a une valeur dans $\langle h(\mathbb{A})\rangle^\sharp$. Si cette valeur est dans I alors elle est aussi dans J. On en déduit directement que pour tout p, $[\![\mathbf{M}, h, I]\!]^+_p(u) \leq [\![\mathbf{M}, h, I']\!]^+_p(u)$. CQFD

Du lemme 4.26, on déduit directement la décidabilité de la relation de domination.

Théorème 4.27. *Le problème de la domination entre fonctions de coût reconnaissables est décidable.*

Démonstration. En effet, il est toujours possible d'après le lemme 4.23 de considérer que deux fonctions reconnaissables de coût sont reconnues par le même monoïde de stabilisation $\mathbf{M}$ et la même application h. Il suffit donc de conclure en appliquant le lemme 4.26, le calcul de $\langle h(\mathbb{A})\rangle^\sharp$ pouvant être effectué par un simple processus de saturation. CQFD

Une autre conséquence du lemme 4.26 nous sera utile pour présenter le monoïde de stabilisation syntaxique.

Proposition 4.28. *Considérons deux fonctions de coût reconnue par* $\mathbf{M}, h, I$ *et* $\mathbf{M}', h', I'$ *respectivement, alors les deux énoncés suivants sont équivalents :*

– $[\![\mathbf{M}, h, I]\!] \preccurlyeq [\![\mathbf{M}', h', I']\!]$,
– *pour toute* $\sharp$*-expression* E *sur* $\mathbb{A}$,

$$\textit{valeur}(h(E)) \in I \quad \textit{implique} \quad \textit{valeur}(h'(E)) \in I' \ ,$$

où $h(E)$ *(resp.* $h'(E)$*) dénote la* $\sharp$*-expression sur* M *(resp. sur* M'*) obtenue de* E *en substituant* $h(a)$ *(resp.* $h'(a)$*) à chaque lettre* $a \in \mathbb{A}$.

Démonstration. D'après le lemme 4.23, il existe un monoïde de stabilisation $\mathbf{M}''$, une application h'' de $\mathbb{A}$ dans $\mathbf{M}''$ et deux morphismes de monoïdes de stabilisation f, f', de $\mathbf{M}''$ dans $\mathbf{M}$ et $\mathbf{M}'$, respectivement tels que $h = f \circ h''$ et $h' = f' \circ h''$. De plus, $[\![\mathbf{M}'', h'', f^{-1}(I)]\!] = [\![\mathbf{M}, h, I]\!]$ et $[\![\mathbf{M}'', h'', f'^{-1}(I')]\!] = [\![\mathbf{M}, h, I']\!]$.

Il nous suffit donc de montrer que l'énoncé $(\star)$:

pour toute $\sharp$-expression E sur $\mathbb{A}$, $\textit{valeur}(h(E)) \in I$ implique $\textit{valeur}(h'(E)) \in I'$.

est équivalent au fait $(\star\star)$:

$\langle h''(\mathbb{A})\rangle^\sharp \cap f^{-1}(I) \subseteq f'^{-1}(I')$,

pour pouvoir appliquer le lemme 4.26 et conclure.

Supposons $(\star\star)$. Soit E une $\sharp$-expression sur $\mathbb{A}$ telle que $\textit{valeur}(h(E)) \in I$. Comme $f(\textit{valeur}(h''(E))) = \textit{valeur}(h(E)) \in I$, nous avons $\textit{valeur}(h''(E)) \in f^{-1}(I)$. Comme de plus $\textit{valeur}(h''(E)) \in \langle h''(\mathbb{A})\rangle^\sharp$, nous pouvons déduire que $\textit{valeur}(h''(E)) \in f'^{-1}(I')$. D'où $\textit{valeur}(h'(E)) = f'(\textit{valeur}(h''(E))) \in f'(f'^{-1}(I')) \subseteq I'$. La propriété $(\star)$ est établie.

La réciproque est similaire : il s'agit de montrer (contraposée), qu'en supposant $\textit{valeur}(h'(E)) \notin I'$, il est possible de déduire que $\textit{valeur}(h'(E)) \notin I$. Le même enchainement de déductions s'applique. CQFD

4.7 Clôture sous inf-projection et sup-projection

Rappelons qu'étant donné un morphisme lettre à lettre z de $\mathbb{A}^*$ dans $\mathbb{B}^*$, et une fonction de coût f sur l'alphabet $\mathbb{A}$ reconnue par $\mathbf{M}, h, I$, notre objectif est reconnaître la fonction de coût qui qui a tout mot $u \in \mathbb{B}^*$:

$$f_{\inf,z}(u) = \inf\{f(v) \ : \ z(v) = u\} \ .$$

Ainsi, il s'agit de considérer toutes les images inverses de u par v, et de minimiser le résultat par f sur ses entrées.

L'inf-projection nécessite (comme pour les monoïdes standards) une construction des parties. En effet, il s'agit de considérer, pour toutes les images inverses d'un mot, quelles pourraient être les valeurs de calculs sur ces entrées. La construction doit donc maintenir un sous-ensemble du monoïde de départ. Dans notre cadre ordonné, il s'agit de considérer

les idéaux. Soit $\downarrow(\mathbf{M})$ l'ensemble des idéaux sur $\mathbf{M}$. Nous équipons $\downarrow(\mathbf{M})$ d'une structure d'ordre par :

$$I \leq J \qquad \text{si} \qquad I \subseteq J \ ,$$

d'un produit par :

$$A \cdot B \quad = \quad \{a \cdot b \ : \ a \in A, \ b \in B\}\downarrow \ ,$$

et d'un opérateur de stabilisation par :

$$E^\sharp = \langle E \rangle^{\sharp +}\downarrow \ .$$

Lemme 4.29 ([28]). *$M_\downarrow = \langle \downarrow(M), \cdot, \sharp, \subseteq \rangle$ est un monoïde de stabilisation. De plus, si $\mathbf{M}, h, I$ reconnait une fonction de coût f sur l'alphabet $\mathbb{A}$, et z est une application de $\mathbb{A}$ dans $\mathbb{B}$, alors*

$$M_\downarrow, h', \{A \in M_\downarrow \ : \ A \subseteq I\} \ .$$

reconnait $f_{\inf,z}$, où $h'(b) = h(z^{-1}(b))\downarrow$ pour tout b de $\mathbb{B}$.

La preuve de cet énoncé (voir [28]) s'obtient en comparant convenablement les calculs dans le monoïde $\mathbf{M}$ au calculs dans le monoïde $\mathbf{M}_\downarrow$. Une construction similaire permet d'obtenir la clôture sous sup-projection.

Lemme 4.30 ([28]). *Les fonctions de coût reconnaissables sur les mots finis sont clos sous* sup*-projection.*

4.8 Le cas des mots infinis

Nous avons présenté jusqu'à présent le cas des mots finis. La situation sur les mots infinis n'est pas très différente. Elle n'a pas fait l'objet de publications ce jour.

Les **algèbres de Wilke** (introduites dans [107]) permettent une description des langages réguliers de mots infinis au moyen d'une structure algébrique finie. Le lecteur peut se référer pour plus d'information à [23] par exemple. Nous décrivons dans cette section les algèbres de Wilke de stabilisation. Il s'agit de la combinaison naturelle des monoïdes de stabilisation avec les algèbres de Wilke. L'objet produit possède les propriétés attendues, et en particulier nous permet de donner une notion de fonctions reconnaissable de coût sur les mots finis qui se trouvent coïncider, comme dans le cas des mots finis, avec les fonctions de coût définissables en logique monadique de coût.

Ainsi, un **algèbre de Wilke de stabilisation**

$$\langle S, S_\infty, \cdot, \sharp, \omega, \leq \rangle$$

est telle que :

- S et S_∞ sont deux ensembles finis disjoints, correspondants aux mots finis non-vide (pour S) et aux mots infinis (pour S_∞),
- l'opérateur binaire $\cdot$ est une opération associative de $S \times S$ dans S et de $S \times S_\infty$ dans S_∞, *c.-à-d.* que pour tous éléments a, b de S et c dans $S \uplus S_\infty$,

$$(a \cdot b) \cdot c = a \cdot (b \cdot c) \ ,$$

- l'opérateur unaire $\sharp$ est une application de $E(S)$ dans $E(S)$, telle que
  - $(a \cdot b)^\sharp \cdot a = a \cdot (b \cdot a)^\sharp$,
- l'opérateur unaire ω est une application de $E(S)$ dans S_∞, telle que
  - $(a \cdot b)^\omega = a \cdot (b \cdot a)^\omega$,
  - $(e^\sharp)^\omega = e^\omega$,
- la relation $\leq$ est à la fois un ordre sur S et un ordre sur S_∞, compatible avec toutes les opérations, *c.-à-d.*,
  - pour tous éléments $a \leq a'$ de S et tous $b \leq b'$ dans S ou S_∞, $a \cdot b \leq a' \cdot b'$,
  - $e^\sharp \leq f^\sharp$ pour tous idempotents $e \leq f$,
  - et $e^\omega \leq f^\omega$ pour tous idempotents $e \leq f$,
- pour tout $e \in E(S)$, $e^\sharp \leq e$.

On reconnaît dans cette définition la combinaison naturelle des monoïdes de stabilisation avec les algèbres de Wilke, à cela près que l'opération ω n'est définie que sur les idempotents (contrairement à la définition originelle des algèbres de Wilke). Il s'agit d'un choix de nature technique. La seule chose nouvelle est l'apparition de la règle $(e^\sharp)^\omega = e^\omega$, qui est assez naturelle.

Continuer l'extension de la théorie requiert de définir ce que sont les n-calculs/n-sous-calculs/n-sur-calculs dans le cadre des algèbres de Wilke syntaxiques. Appelons les **ω-n-calculs**/**ω-n-sur-calculs**/**ω-n-sous-calculs**. Là encore, cette définition est naturelle : il s'agit d'arbres finis ou infinis, dans lesquels un nouveau type de nœud x est possible : les **nœuds d'ω-itérations**, ayant une infinité de fils étiquetés $y_1, y_2, \ldots$ (de gauche à droite), et tels que la valeur des nœuds $y_1, \ldots$ est égale au même idempotent e, et la valeur de x est e^ω (*resp.* supérieure à e^ω pour un sur-calcul, *resp.* inférieure à e^ω pour un sous-calcul).

Notons la contrainte syntaxique qu'un arbre d'ω-calcul possède exactement un nœud d'ω-iteration, et que celui-ci ne peut apparaitre dans le membre gauche d'un nœud de produit, ou sous un nœud de stabilisation ou d'un nœud idempotent.

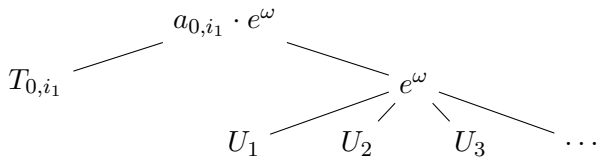
Les résultats fondamentaux des calculs restent valides, à commencer par l'existence d'un calcul. En fait, tout se déduit du cas fini.

Lemme 4.31. *Pour toute algèbre de Wilke de stabilisation $\langle S, S_\infty, \cdot, \sharp, \omega, \leq \rangle$ tout mot infini $u \in S^\omega$ et tout entier n, il existe un ω-calcul sur u de hauteur au plus*[3] $4|S| + 2$.

Démonstration. Soit $u = a_0 a_1 \ldots$ Pour tous $i < j$, soit $a_{i,j}$ une valeur de S telle que $a_i a_{i+1} \ldots a_{j-1}$ admette un n-calcul $T_{i,j}$ de hauteur au plus $3|S|$ de valeur $a_{i,j}$ (c'est possible

3. Il est possible en fait d'obtenir une meilleur borne, 3|S|-1, mais l'avantage de cet énoncé est qu'il s'obtient directement en appliquant le résultat sur les mots finis.

d'après le théorème 4.10). Par le théorème de Ramsey, il existe b dans S et des entiers $0 < i_1 < i_2 < \dots$ tels que $a_{i_l,i_{l+1}} = b$ pour tout l. On sait qu'il existe $m \leq n$ tel que $b^m = e$ est un idempotent. Ainsi chaque $a_{i_l} \dots a_{i_{l+m}-1}$ possède un n-calcul de hauteur au plus $3|S| + m \leq 4|S|$ de valeur e. Soit U_l cet arbre. Construisons alors l'ω-n-calcul suivant :



Il s'agit d'un ω-n-calcul de hauteur au plus $4|S|$ sur le mot u. CQFD

Le résultat permettant de comparer les calculs, pendant du théorème 4.15, se dérive également du cas des mots finis.

Théorème 4.32. *Pour tout entier p, il existe un polynôme $\alpha : \mathbb{N} \to \mathbb{N}$ tel que pour tout ω-$\alpha(n)$-sur-calcul de valeur b sur un mot infini u et tout ω-n-sous-calcul sur le même mot infini de valeur a de hauteur au plus p, alors*

$$a \leq b \ .$$

Démonstration. La démonstration s'obtient en utilisant le cas des mots finis en combinaison avec les techniques habituelles pour les algèbres de Wilke. Nous ne la développons pas plus dans ce document. CQFD

Tous les autres résultats concernant les monoïdes de stabilisation s'étendent alors au cas des mots infinis. Il s'agit en particulier de l'existence d'une notion de fonction de coût reconnaissable sur les mots infinis, de la clôture des fonctions de coût reconnaissables sur les mots infinis sous min, max, inf-projection et sup-projection, l'équivalence avec la logique monadique de coût, ainsi que la décidabilité de la domination. Les preuves sont identiques. Nous ne développons pas plus avant ce cas.

Chapitre 5

Les automates à coût

Nous avons vu au cours du chapitre précédent comment la notion de fonction régulière de coût pouvait être introduite en utilisant la notion de reconnaissabilité par monoïde de stabilisation. Nous présentons une seconde approche utilisant une forme adaptée d'automates, les automates à coûts.

Ces automates permettent de définir des fonctions des mots dans $\mathbb{N} \cup \{\infty\}$. Nous verrons qu'ils définissent les mêmes fonctions de coûts que la logique monadique de coût ou les monoïdes de stabilisation. Si historiquement ils sont apparus comme sujet d'étude principal (les automates de distance de Hashiguchi, puis les formes d'automates étudiés par Bala et surtout Kirsten), ils ne sont en fait pas les objets d'étude les plus pertinents dans le cas des mots. Ils deviennent inévitables dans le cas des arbres.

Nous présentons dans ce chapitre les automates à coût dans le cadre des mots. Cela nous permet de fixer une partie de la terminologie, bien qu'aucun résultat de décidabilité nouveau n'en sera déduit. La dynamique de ces notions apparaitra plus clairement dans les chapitres suivants avec l'étude des jeux, et les automates d'arbres.

Les automates à coût, dans leur forme non-déterministe, se déclinent essentiellement en deux variantes : les `B`-automates et les `S`-automates. Ces deux formes se distinguent en ce que le non-déterminisme est interprété comme calculant un minimum pour les `B`-automates et comme un maximum dans le cadre des `S`-automates. Ces définitions procurent une nature duale de ces deux classes, et bien entendu conditionnent leurs propriétés. S'il convient, dans l'absolu, de développer les deux notions, nous nous attardons plus longuement dans ce document sur les `B`-automates. Bien entendu les deux notions seront réconciliées quand nous considérerons les modèles alternants d'automates et la théorie des jeux. Nous attacherons une attention plus particulière au modèle des `B`-automates dans ce chapitre.

Le reste de ce chapitre se décompose comme suit. Les `B`-automates sont introduits au cours de la section 5.1. De premiers résultats élémentaires sont présentés au cours de la section 5.2. Plusieurs variantes du modèle sont considérés à la section 5.3. Le modèle des `B`-automates hiérarchiques, qui joue un rôle important par la suite, sera présenté à la

section 5.4. Nous présenterons à la section 5.5 les expressions B-régulières et montrerons leur équivalence avec les B-automates. Le modèle dual des S-automates est le sujet de la section 5.6. Les résultats centraux d'équivalence seront présentés et commentés à la section 5.7. Enfin le cas des mots infinis sera brièvement mentionné à la section 5.8 (il sera développé au cours de la partie suivante).

5.1 Les B-automates

Les B-automates sont des automates finis qui à chaque mot associent une valeur dans $\mathbb{N} \cup \{\infty\}$. Ils utilisent des compteurs qu'ils peuvent incrémenter, remettre à zéro, et observer. La fonction calculée par un tel automate est le minimum sur toutes les exécutions de la plus grande valeur observée (quel que soit le compteur observé). Les automates de distance en sont un cas particulier.

Formellement, un B-*automate* $\mathcal{A} = \langle Q, \mathbb{A}, I, F, \Gamma, \Delta \rangle$ est décrit par :

- un ensemble fini d'*états* Q,
- un **alphabet** fini $\mathbb{A}$,
- un ensemble d'états **initiaux** $I \subseteq Q$,
- un ensemble d'états **finaux** $F \subseteq Q$,
- une relation de transition $\Delta \subseteq Q \times \mathbb{A} \times A_\Gamma \times Q$, où $A_\Gamma = \{\epsilon, \mathtt{ic}, \mathtt{r}\}^\Gamma$.

Un B-automate sera dit **déterministe** si la relation Δ est une fonction de $Q \times \mathbb{A}$ dans $A_\Gamma \times Q$ et il n'y a qu'un seul état initial. Un B-automate déterministe sera dit **complet** si de plus δ est totalement définie. Une **exécution** σ de $\mathcal{A}$ sur un mot $a_1 \dots a_n \in \mathbb{A}^*$ est une séquence

$$\sigma = (p_0, a_1, t_1, p_1) \dots (p_{n-1}, a_n, t_n, p_n)$$

telle que $(p_{i-1}, a_i, t_i, p_i) \in \Delta$ pour tout $i = 1 \dots n$. Si de plus $p_0 \in I$, $p_n \in F$, l'exécution est dite **acceptante**.

Une exécution induit pour chaque compteur un mot sur $\{\mathtt{i}, \mathtt{c}, \mathtt{r}\}^*$. Les lettres $\mathtt{i}, \mathtt{c}, \mathtt{r}$ sont appelées **actions élémentaires**. Au début de l'exécution, les compteurs partagent la même valeur 0, puis évoluent suivant les règles suivantes :

Incrément : si la lettre est `i` (`i` pour «increment»), la valeur du compteur est augmentée de 1,

Remise à 0 : si la lettre est `r` (`r` pour «reset»), la valeur du compteur est remise à 0,

Observation : si la lettre est `c` (`c` pour «check»), la valeur du compteur ne change pas, mais est **observée**.

Nous définissons $C(u)$ comme l'ensemble des valeurs observées au cours de u. Nous utiliserons la notation $C(u)$ quand u est une séquence d'actions élémentaires, $u \in \{\epsilon, \mathtt{ic}, \mathtt{r}\}^*$, quand u est une séquence de tuples d'actions (non nécessairement élémentaires) indexé par un ensemble de compteurs, *c.-à-d.* $u \in (\{\epsilon, \mathtt{ic}, \mathtt{r}\}^*)^\Gamma$, ou quand u est tout simplement une exécution. Remarquons que dans le cas de plusieurs compteurs, $C(u)$ collecte toutes les valeurs observées sans tenir compte du compteur qui en est à l'origine.

Le **coût** de la séquence, pour tous les cas mentionnés ci-dessus, est alors :

$$coût_B(u) = \sup C(u)$$

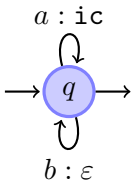
Rappelons que $\sup \emptyset = 0$ est un cas particulier. Cela signifie que si aucun compteur n'est observé, le coût est 0. Cela est en particulier le cas quand il n'y a pas de compteurs.

Il est maintenant très simple de définir la sémantique d'un B-automate On définit la valeur d'un mot $u \in \mathbb{A}^*$ pour $\mathcal{A}$ comme :

$$[\![\mathcal{A}]\!]_{\mathtt{B}}(u) = \inf\{coût_B(\sigma) \ : \ \sigma \text{ exécution acceptante de } \mathcal{A} \text{ sur } u\} \ .$$

Rappelons que $\inf \emptyset = \infty$. Cela a pour conséquence que s'il n'existe aucune exécution acceptante sur u, alors $[\![\mathcal{A}]\!]_{\mathtt{B}}(u) = \infty$.

Exemple 5.1. Le B-automate suivant compte le nombre d'occurrences de la lettre a.



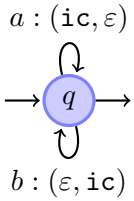
On utilise les conventions suivantes. Les états sont représentés par des cercles. Les états initiaux ont une flèche entrante sans origine, et les états finaux sont origines d'une flèche sans destination. Une transition (p, a, t, q) est représentée comme une flèche d'origine p, de destination q, étiquetée par $a : t$. Dans cet exemple, comme il n'y a qu'un compteur, t est représenté comme l'action sur l'unique compteur.

Remarque 5.2. Si $\mathcal{A}$ ne possède aucun compteur (*c.-à-d.* $\Gamma = \emptyset$) alors $\mathcal{A}$ peut être vu comme un automate non-déterministe acceptant le langage régulier $L = \mathcal{L}(\mathcal{A})$. On a alors pour tout mot u :

$$[\![\mathcal{A}]\!](u) = \chi_L(u) = \begin{cases} 0 & \text{si } u \in L, \\ \infty & \text{sinon.} \end{cases}$$

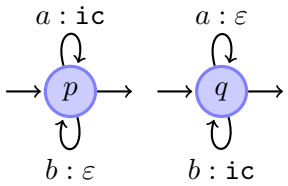
En particulier, si $\mathcal{A}$ et $\mathcal{A}'$ sont des B-automates sans compteurs, alors $\mathcal{L}(\mathcal{A}) \subseteq \mathcal{L}(\mathcal{A}')$ ssi $[\![\mathcal{A}]\!] \geq [\![\mathcal{A}']\!]$.

Exemple 5.3. Le B-automate suivant calcule le maximum du nombre d'occurrences de la lettre a et du nombre d'occurrences de la lettre b. Il utilise deux compteurs.



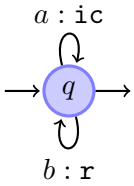
Cette fois-ci, deux compteurs sont utilisés, et les actions sont donc représentées comme une paire formée de l'action sur le premier compteur, suivie de l'action sur le deuxième compteur.

Exemple 5.4. Le B-automate à un compteur suivant calcule le minimum du nombre d'occurrences de la lettre a et du nombre d'occurrences de la lettre b.

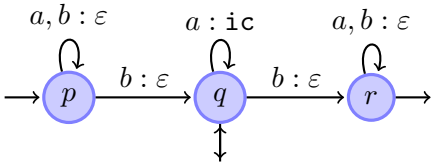


Cette fois-ci, l'automate doit deviner, en utilisant le non-déterminisme, s'il compte le nombre d'occurrences de a ou le nombre d'occurrences de b.

Exemple 5.5. Le B-automate suivant calcule la longueur maximale d'un segment formé uniquement de a.



Exemple 5.6. Le B-automate suivant calcule la fonction minseg qui a chaque mot associe la longueur minimale d'un segment formé uniquement de a, *c.-à-d.* $\text{minseg}(a^{n_0}b \dots ba^{n_k}) = \min(n_0, \dots, n_k)$.



Nous utilisons ici l'abréviation $a, b : t$ annotant une flèche d'origine p et de destination q, pour dénoter qu'il existe les deux transitions (p, a, t, q) et (p, b, t, q). Cet automate utilise son non-déterminisme, pour deviner le segment de a qu'il s'agit de mesurer.

5.2 Résultats élémentaires

Nos premiers résultats relient le monde des B-automates à celui de la logique monadique de coût du chapitre précédent.

Proposition 5.7. *Les fonctions définissables par* B*-automate sont définissables en logique monadique de coût.*

Idée. Soit $\mathcal{A}$ un B-automate. Par définition, $[\![\mathcal{A}]\!]_{\mathtt{B}}(u) \leq N$ si et seulement s'il existe une exécution acceptante de l'automate sur u telle que pour tout compteur, il n'y a pas plus de N incréments de ce compteur consécutifs. Cette énoncé se formalise directement en logique monadique de coût en utilisant les codages habituels. CQFD

Nous établissons également dans cette section certains résultats de clôture sur les fonctions calculées par B-automates.

Proposition 5.8. *Le support d'une fonction définie par* B*-automate est un langage régulier.*

Idée. Une première méthode consiste à traduire l'automate en logique monadique de coût en utilisant la proposition 5.7, puis en appliquant le proposition 2.8 qui montre que le support d'une fonction définie en logique monadique de coût est régulier.

Une construction directe est également possible. Il suffit d'enlever les actions sur les compteurs pour obtenir un automate non-déterministe acceptant un langage. Ce langage est le support de la fonction calculée par l'automate de départ. CQFD

Une forme de réciproque est aussi vraie.

Proposition 5.9. *Les fonctions caractéristiques des langages rationnels de mots sont acceptées par* B*-automates.*

Idée. Considérons un automate de mot non-déterministe acceptant un langage L. Cet automate peut être vu comme un B-automate sans aucun compteur. La fonction acceptée par ce B-automate est alors exactement la fonction caractéristique de L. CQFD

Proposition 5.10. *Étant donnée f calculée par un* B*-automate et un entier positif n, l'ensemble des mots de valeur n est régulier.*

Idée. Il suffit de convertir l'automate en logique monadique de coût en utilisant la proposition 5.7, de calculer son support au moyen la proposition 2.7. Une autre construction consiste à construire un automate qui dans ses états «compte jusqu'à n». Un automate fini peut donc simuler un B-automate pour toutes valeurs des compteurs inférieures à n. Sa taille est celle de l'automate originale multipliée par $(n+1)^k$, où k est le nombre de compteurs. CQFD

Les résultats suivants montrent la clôture des fonctions calculées par B-automates sous min (proposition 5.11), max (proposition 5.12) et min-projection et inf-projection. Ces constructions correspondent directement à la clôture sous union et intersection et projection des langages rationnels.

Proposition 5.11. *Les fonctions calculées par* B*-automate sont closes sous* min.

Idée. Il suffit d'effectuer l'union disjointe des deux automates. L'infimum sur toutes les exécutions dans la définition de la sémantique de l'automate entraîne en particulier que le chemin appartenant à l'automate donnant la plus petite valeur est sélectionné. À titre d'exemple, on peut observer le rapport entre l'automate de l'exemple 5.1 et l'automate de l'exemple 5.4. CQFD

Proposition 5.12. *Les fonctions calculées par* B*-automate sont closes sous* max*.*

Idée. Il s'agit cette fois-ci de construire un automate produit travaillant sur l'union disjointe des compteurs. Cet automate simule une exécution des deux automates d'origine, effectuant les manipulations des compteurs sur la copie des compteurs correspondante. À titre d'exemple, on peut observer le rapport entre l'automate de l'exemple 5.1 et l'automate de l'exemple 5.3. CQFD

Proposition 5.13. *Les fonctions calculées par* B*-automate sont closes sous* inf*-projection.*

Idée. Soit z un morphisme lettre à lettre de $\mathbb{A}^*$ dans $\mathbb{B}^*$, et $\mathcal{A}$ un B-automate sur l'alphabet $\mathbb{A}$. Il s'agit de construire un automate sur l'alphabet $\mathbb{B}$, qui accepte la fonction

$$[\![\mathcal{A}']\!]_{\mathtt{B}}(u) = \min\{[\![\mathcal{A}]\!]_{\mathtt{B}}(v) \ : \ z(v) = u\} \ .$$

Pour cela, il suffit de partir de l'automate $\mathcal{A}$ et de remplacer chaque transition de la forme (p, a, t, q) par la transition $(p, f(a), t, q)$. Ainsi, le nouvel automate utilise son non-déterminisme pour d'une part deviner un antécédent du mot en entrée pour z, et d'autre part deviner une exécution de l'automate de départ sur cet antécédent. CQFD

En revanche, les fonctions calculées par B-automates ne sont pas closes sous sup-projection. Nous verrons qu'elles le sont, mais seulement à $\approx$-près.

5.3 Variantes du modèle des B-automates

Nous avons vu ci-dessus le modèle générique des B-automates. En fait, la situation est comparable par certains aspects à ce qui se passe pour les langages de mots infinis. Dans ce cas les automates peuvent-être déterministes, non-déterministes, alternants, et utiliser des conditions de Büchi, de Rabin, de Streett, ou de Müller, et toutes les combinaisons (sauf le cas déterministe Büchi) ont le même pouvoir d'expression. Il est aussi possible d'autoriser ou non les ε-transition, d'autoriser l'automate à rebrousser chemin, etc... L'une des grandes richesses de la théorie des automates est la capacité de passer d'un modèle à l'autre. Nous commentons dans cette section plusieurs variantes du modèle : les *automates de distance*, puis les B*-automates déterministes* et enfin *les* B*-automates avec ϵ-transitions.*

Les automates de distance

Les automates de distance, introduits par Hashiguchi [42] sont les premiers automates à avoir été considérés (voir section 1.2). Un automate de distance est un automate pondéré sur le semi-anneau tropical $((\mathbb{N} \cup \{\infty\}, \min, +))$. Cela revient dans le formalisme des B-automates à définir un **automate de distance** comme un B-automate à 1 compteur, sans remise à 0. Ainsi, un automate de distance ne possède qu'un unique compteur, sur lequel les actions possibles sont ε et `ic` seulement.

En termes de fonctions régulières de coût, les automates de distance sont strictement plus faibles que les B-automates.

Proposition 5.14. *La fonction* minseg *qui a chaque* $a^{n_0}b \dots ba^{n_k}$ *associe* $\max(n_0, \dots, n_k)$ *est acceptée par un* B*-automate mais par aucun automate de distance, même modulo* $\approx$.

Dans le cas des automates de distance, il est possible d'être plus précis.

Proposition 5.15. *Une fonction des mots dans* $\mathbb{N} \cup \{\infty\}$ *est définissable par automate de distance si et seulement si elle est définissable dans le fragment de distance, c.-à-d. par une formule de la forme*

$$\exists X \ |X| \leq N \wedge \psi(X) \ ,$$

où $\psi(X)$ *est monadique.*

Idée. Des automates vers la formules. La formule de distance commence par deviner l'ensemble X des positions pour lesquelles l'automate effectue un incrément, et vérifie que cet ensemble est de taille au plus N. La formule ψ devine alors une exécution de l'automate sur l'entrée, et vérifie que cette exécution est valide, acceptante, et qu'elle n'effectue des incréments qu'au cours des transitions correspondantes aux positions dans X.

Dans le sens contraire, on sait que $\psi(X)$ peut être transformé en un automate de mot qui étant donné u et X accepte le mot $\langle u, X\rangle$ si et seulement si $u \models \psi(X)$. Il suffit alors de voir cet automate comme un automate non déterministe sur u qui devine X et effectue un incrément si et seulement si la transition correspond à une position dans X. CQFD

B-automates déterministes

Dans cette section nous entamons la description des différentes variations possibles autour de la notion de B-automate. Ces équivalences ne sont valables que modulo $\approx$ (certaines sont valables exactement).

Notre premier résultat montre que la situation est différente du cas des langages de mots. En effet, nous commençons en montrant que, même considérés modulo $\approx$, les B-automates ne peuvent pas être rendus déterministes.

Proposition 5.16. *La fonction* $a^m b^n \mapsto \min(m, n)$ *ne peut être reconnue par un* B*-automate déterministe modulo* $\approx$.

Démonstration. Par la contraposée, supposons qu'il existe une fonction de correction α et un B-automate déterministe $\mathcal{A}$ qui définisse une fonction $\approx_\alpha$-équivalente à $a^m b^n \mapsto \min(m,n)$. Notons $\delta(u)$ l'état atteint à partir de l'état initial en lisant le mot u en entrée. Nous notons dans cette preuve par $\delta(u)$ l'unique état atteint par l'automate après avoir lu le mot u.

Commençons par analyser la structure de l'automate. Remarquons tout d'abord que le mot $a^m b^n$ n'a pas la valeur ∞, et donc $\delta(a^m b^n)$ existe pour tous m, n, et est final. Comme l'ensemble d'états est fini, il existe k, l tels que $\delta(a^k) = \delta(a^{k+l})$. De même, il existe k', l', tels que $\delta(a^k b^{k'}) = \delta(a^k b^{k'+l'})$.

Supposons qu'il existe un compteur α tel que sur la boucle parcourue en lisant a^l a partir de l'état $\delta(a^k)$, γ est incrémenté, mais n'est pas remis à 0. Nous obtenons alors que $[\![\mathcal{A}]\!](a^{k+nl}) \geq n$, et donc $[\![\mathcal{A}]\!]$ n'est pas borné sur a^{k+nl}, alors que $a^n b^m \mapsto \min(m,n)$ l'est. Contradiction. De même, il est impossible qu'un compteur soit incrémenté mais non remis à 0 lorsque l'on parcours la boucle d'origine $\delta(a^k b^{k'})$ obtenue par lecture du mot $b^{l'}$. Ainsi, nous savons que tous les compteurs sont soit remis à 0, soit jamais incrémentés dans ces boucles. Nous en déduisons que $[\![\mathcal{A}]\!](a^{k+nl} b^{k'+nl'}) \leq k+l+k'+l'$, et donc $[\![\mathcal{A}]\!]$ est borné sur $\{a^{k+nl} b^{k'+nl'} \ : \ n \in N\}$, alors que $a^n b^m \mapsto \min(m,n)$ ne l'est pas. Il s'agit de nouveau d'une contradiction. CQFD

Ce résultat est, *a priori*, très négatif, et montre qu'il est délicat de travailler avec les B-automates. En fait, la situation possèdent une certaine ressemblance avec d'autres modèles d'automates, comme en particulier les automates de Büchi sur les mots infinis ou les automates de parité sur les arbres infinis. Dans ces deux cas, il est impossible de déterminiser les automates. Différentes techniques sont donc utilisées pour contourner cette difficulté, et en particulier, pour montrer la clôture sous complément [16, 87].

Nous verrons en particulier que, bien que les B-automates déterministes soient strictement moins expressifs que les B-automates en général, il est possible de définir une *notion sémantique*, le *déterminisme en histoire* qui peut être utilisée à la place du déterminisme dans certaines situations. Contrairement au résultat précédent, tout B-automate peut être transformé en un B-automate déterministe en histoire équivalent. Cette technique s'avérera cruciale pour traiter le cas des arbres. Elle sera développée à la section 7.7.

Automates avec ϵ-transitions

L'un des premiers résultats que l'on apprend en théorie des langages est la possibilité d'éliminer les ϵ-transitions d'un automate non-déterministe de mots. Ce résultat reste valable dans le cas des automates à coût.

Un **B-automate avec ϵ-transition** est un B-automate sur l'alphabet $A \cup \{\epsilon\}$. Étant donné $u \in A \cup \{\epsilon\}$, $\overline{u}^\epsilon$ est le mot de A^* obtenu de u en éliminant toutes les occurrences de la lettre ϵ. On pose pour tout $u \in A^*$:

$$[\![\mathcal{A}]\!]^\epsilon(u) = \inf\{[\![\mathcal{A}]\!](v) \ : \ \overline{v}^\epsilon = u\} \ .$$

On remarque que dans le cas sans compteurs, cette définition est consistante avec la notion habituelle d'automates avec ϵ-transitions.

L'objectif est d'établir la proposition suivante.

Proposition 5.17. *Les* B*-automates avec ϵ-transitions sont $\approx$-équivalents aux* B*-automates.*

Afin d'éliminer les ϵ-transitions, le point clef est de compresser les actions se déroulant sur une séquence de transitions en une seule action. Il s'agit donc de pouvoir effectuer le produit d'actions. Pour cette raison, la construction brièvement décrite ci-dessus peut-elle être interprétée en termes de monoïdes des stabilisation.

Ainsi, définissons l'opération de produit $\cdot$ sur $\{\varepsilon, \mathtt{ic}, \mathtt{r}\}$ comme le maximum pour l'ordre $\varepsilon < \mathtt{ic} < \mathtt{r}$ (il s'agit du produit du monoïde de stabilisation reconnaissant $coût_B$). Ce produit est étendu composante par composante à $A_\Gamma = \{\varepsilon, \mathtt{ic}, \mathtt{r}\}^\Gamma$. Soit également $\pi : \{\varepsilon, \mathtt{ic}, \mathtt{r}\}^* \to \{\epsilon, \mathtt{ic}, \mathtt{r}\}$ l'extension du produit à un mot de longueur quelconque $u \in A_\Gamma^*$ (avec $\pi(\varepsilon) = \epsilon$).

L'idée de la construction permettant d'éliminer les ϵ-transitions est alors très simple. Le nouvel automate contient les états de l'automate d'origine, et il y a une transition $(p, a, \pi(v), q)$ dans le nouvel automate si dans le premier il existe un chemin étiqueté par $\epsilon^* a \epsilon$ en entrée, et par v en sortie. De plus, un nouvel état isolé, à la fois initial et final est ajouté si l'automate d'origine attribue une valeur finie au mot vide.

Cette construction n'est valable modulo $\approx$. En effet, un automate à ϵ-transitions a la possibilité d'effectuer, par exemple, deux actions `ic` consécutives à chaque fois qu'une lettre est lue. L'automate après la transformation n'effectuera plus qu'une seule de ces actions. Ainsi le résultat sera divisé par 2. Cette erreur est acceptable quand elle est considérée modulo $\approx$.

5.4 Les B-automates hiérarchiques

Une sous-classe particulièrement importante des B-automates sont ceux possédant la propriété d'être hiérarchiques, les hB-automates. La situation est comparable à celle des automates de mots infinis. En effet, les B-automates hiérarchiques sont aux B-automates ce que les automates à parité sont aux automates de Streett. Dans cette section nous ne faisons que présenter cette notion et montrer son équivalence avec les B-automates non-hiérarchiques. Elle nous servira dans la suite pour montrer l'équivalence avec les expressions régulières. Ce n'est pourtant pas là que la notion prend tout son sens, mais dans le cadre des jeux (théorème 7.12) car, tout comme dans la condition de parité dans les jeux infinitaires, cette condition permet la détermination positionnelle.

Historiquement, les B-automates hiérarchiques ont été introduits avant, les B-automates, par Kirsten [52]. Les B-automates ont été décrits indépendamment dans [7], à la fois dans leur forme générale et dans leur forme hiérarchique. Les constructions de ce chapitre proviennent de [8].

Dans les B-automates hiérarchiques, l'utilisation des compteurs est sujette à restriction : les compteurs sont totalement ordonnés, et incrémenter ou remettre à zéro un compteur réinitialise obligatoirement tous les compteurs inférieurs. Formellement, un B-automate est dit **hiérarchique** (ou encore **hB-automate**) si ses compteurs sont de la forme $\Gamma = \{1, \dots, k\}$, et les seules actions H_k sur les compteurs sont :

- $\mathtt{R}_\ell$ pour $\ell = 0, \dots, k$, tel que $\mathtt{R}_\ell(\gamma) = \begin{cases} \mathtt{r} & \text{si } \gamma \leq \ell \\ \varepsilon & \text{sinon} \end{cases}$
  (le cas particulier de $\mathtt{R}_0$, même en l'absence d'un compteur 0, dénote l'action ε),
- $\mathtt{IC}_\ell$ pour $\ell = 1, \dots, k$, tel que $\mathtt{IC}_\ell(\gamma) = \begin{cases} \mathtt{r} & \text{si } \gamma < \ell \\ \mathtt{ic} & \text{si } \gamma = \ell \\ \varepsilon & \text{sinon.} \end{cases}$

Remarque 5.18. L'une des raisons des bonnes propriétés des conditions hiérarchiques est que le monoïde de stabilisation correspondant est particulièrement simple. Ainsi, considérons l'application $co\hat{u}t_{\mathtt{hB}}$ qui à une séquence

$$u \in \{\mathtt{R}_0, \mathtt{IC}_1, \dots, \mathtt{IC}_k, \mathtt{r}_k\}$$

associe $co\hat{u}t_{\mathtt{B}}$.

Le monoïde de stabilisation correspondant est le suivant (il est aussi décrit précisément par Kirsten, mais en utilisant une autre terminologie [52]).

- Les éléments sont $\{0, \mathtt{R}_0, \mathtt{IC}_1, \dots, \mathtt{IC}_k, \mathtt{r}_k\}$ (assez naturellement chaque lettre est envoyée sur l'élément de même nom dans le monoïde de stabilisation).
- Le produit $\cdot$ est le maximum pour l'ordre

$$\mathtt{R}_0 \sqsubset \mathtt{IC}_1 \sqsubset \mathtt{R}_2 \sqsubset \cdots \sqsubset \mathtt{IC}_k \sqsubset \mathtt{R}_k \ .$$

- La stabilisation est donnée par

$$\mathtt{R}_\ell^\sharp = \mathtt{R}_\ell \quad \text{pour tout} \quad \ell = 0 \dots k \ ,$$
$$\text{et} \quad \mathtt{IC}^\sharp = 0 = 0^\sharp \quad \text{pour tout} \quad \ell = 1 \dots k \ .$$

- Enfin, l'ordre est total :

$$0 < \mathtt{IC}_k < \mathtt{IC}_{k-1} < \cdots < \mathtt{IC}_1 < \mathtt{R}_0 < \mathtt{R}_1 < \cdots < \mathtt{R}_k \ .$$

L'objet résultant est un monoïde de stabilisation qui reconnaît la fonction $co\hat{u}t_{\mathtt{hB}}$ (en utilisant l'idéal $\{0\}$). L'une des caractéristiques remarquables de ce monoïde de stabilisation est d'être totalement ordonné (cela dit, d'autres ordres conviennent). Cette propriété joue un rôle important dans les propriétés des automates correspondant. Si l'on compare ce monoïde de stabilisation à celui pour k compteurs non-hiérarchiques, il est exponentiellement plus succinct, et en particulier il est linéaire dans le nombre de compteurs. Cette autre propriété participe également des bonnes propriétés algorithmiques de des B-automates (par exemple, le problème de la divergence est polynomial alors qu'il est exponentiel pour les B-automates généraux).

Ce qui justifie l'introduction des B-automates hiérarchiques est que les B-automates et les B-automates hiérarchiques sont équivalents. Ce résultat était présent dans [25] et en fait, exprimé différemment, dans [7]. La preuve donnée ici provient de [33]

Théorème 5.19 ([7],[33]). *Les* B*-automates hiérarchiques sont* $\approx$*-équivalents aux* B*-automates. De plus, cette traduction préserve le nombre de compteurs, ainsi que l'éventuel déterminisme de l'automate en entrée.*

La démonstration présente dans [7] utilise une traduction en expression B-rationnelle et procède ensuite de manipulations syntaxiques de ces expressions. La démonstration que nous donnons ici [33], est beaucoup plus simple, a une meilleur complexité, et préserve en plus le déterminisme de l'automate en entrée. Il s'agit en fait de réutiliser une élégante construction classiquement utilisée pour les automates sur les mots et les arbres infinis, l'enregistrement de dernière occurrence («*last appearance record*»). Cette technique permet de transformer les conditions de Müller, de Streett ou de Rabin en conditions de parité. McNaughton fut le premier à considérer ce type de constructions.

Fixons un ensemble de compteurs Γ. La première étape consiste à construire un automate déterministe qui essentiellement lit en entrée une séquence d'action sur les compteurs de Γ, et calcule son coût en utilisant une condition hiérarchisée. Ainsi, nous construisons un automate déterministe LAR_Γ sur l'alphabet A_Γ tel que :

$$LAR_\Gamma \approx coût_{\mathtt{B}} \ , \tag{5.1}$$

c.-à-d. qui accepte la fonction de coût correspondant à la condition d'acceptation de l'automate. Soit $k = |\Gamma|$. Nous définissons LAR_Γ comme suit :

- les états sont des permutations de Γ, *c.-à-d.* des bijections de $\{1, \ldots, k\}$ sur Γ,
- l'alphabet en entrée est A_Γ,
- les compteurs sont $\{1, \ldots, k\}$,
- la transition depuis l'état π, en lisant la lettre $t \in A_\Gamma = \{\varepsilon, \mathtt{ic}, \mathtt{r}\}^\Gamma$ en entrée est construite comme suit. Posons :

$$i = \max\{\ell \ : \ t(\pi(\ell)) = \mathtt{ic}\} \qquad \text{et} \qquad r = \max\{\ell \ : \ t(\pi(\ell)) = \mathtt{r}\}$$

  (i et r ont pour valeur 0 si le maximum s'effectue sur une ensemble vide). La transition issue de π en lisant t en entrée est alors :
  - $(\pi, t, \mathtt{IC}_i, \pi')$ si $i > r$,
  - $(\pi, t, \mathtt{R}_r, \pi')$ sinon (et en particulier, $(\pi, t, \mathtt{R}_0, \pi')$ si $i = r = 0$).

  où π' est obtenu de π en mettant les compteurs γ tels que $t(\gamma) = r$ en tête de la permutation, et en laissant les autres ordres relatifs inchangés.

Afin d'établir (5.1), il s'agit d'observer l'évolution de la position d'un compteur dans les permutations successives produites par une exécution de LAR_Γ. Pour cela, nous parlerons de l'**indice** d'un compteur γ **dans une permutation** π pour signifier l'unique indice i tel que $\pi(i) = \gamma$. Le lemme suivant décrit comment l'indice d'un compteur est susceptible d'évoluer.

Lemme 5.20. *Considérons une transition (π, t, h, π') de LAR_Γ et un compteur γ d'indice i dans π et d'indice i' dans π', alors :*
- *si $t(\gamma) \neq \mathtt{r}$ alors $i \leq i'$,*
- *si $t(\gamma) \neq \mathtt{r}$ alors $i < i'$ si et seulement s'il existe γ' d'indice dans π supérieur à i tel que $t(i') = \mathtt{r}$.*

Démonstration. Au cas par cas. CQFD

Lemme 5.21. *Si u est une séquence d'incréments hiérarchisés de longueur n^k, alors coût$(u) \geq n$.*

Démonstration. Montrons par récurrence sur ℓ que toute séquence d'incréments hiérarchisés ne touchant qu'aux compteurs inférieurs ou égaux à ℓ et de longueur au moins n^ℓ a un coût au moins égal à n. Pour $k = 0$, la propriété est vérifiée car la longueur de la séquence est nécessairement nulle. Sinon, la séquence u s'écrit comme $u_0\mathtt{IC}_k u_1 \dots \mathtt{IC}_k u_m$ ou les u_i's ne contiennent pas d'incréments de k. Si l'un des u_i's a une taille au moins égale à n^{k-1}, l'hypothèse de récurrence permet de conclure. Sinon, $n^k \leq |u| = |u_0| + |u_1| + \dots + |u_\ell| + l \leq n^{k-1}(\ell + 1) - 1 \leq \ell n^{k-1}$. Il s'ensuit que $\ell \geq n$, et la séquence est donc de coût au moins n. CQFD

Nous pouvons maintenant établir la correction de la construction, *c.-à-d.* (5.1).

Lemme 5.22. *Pour tout $u \in A_\Gamma^*$,*

$$coût(u) \approx_\alpha LAR_\Gamma(u)$$

où $\alpha(n) = k(n+1)^k - 1$.

Démonstration. Fixons une exécution ρ de LAR_Γ (sur un mot $u \in A_\Gamma^*$). Soit $n = coût(\rho)$. Cela veut dire qu'il existe un facteur ρ' de ρ et $\ell \in \{1, \dots, k\}$ tel que dans ρ' (a) $\mathtt{IC}_\ell$ est effectué n fois, et (b) toutes les autres actions $\mathtt{IC}_m$ et $\mathtt{R}_m$ s telles que $m < \ell$. D'après le lemme 5.20, le compteur $\pi(m)$ reste donc inchangé dans ρ'. Il s'ensuit que $\pi(m)$ est incrémenté n fois dans ρ'. D'où $coût(u) \geq n$. Ainsi, nous avons prouvé que $LAR_\Gamma(u) \leq coût(u)$.

Réciproquement, supposons que $coût(u) \geq kn^k$, cela veut dire qu'il y a un facteur ρ' de ρ et un compteur γ tel que γ est incrémenté kn^k fois dans ρ', sans jamais être remis à 0. D'après le Lemme 5.20, l'indice de γ ne peut qu'augmenter dans ρ'. Cela signifie qu'il existe un facteur ρ'' de ρ' dans lequel γ conserve le même indice en étant incrémenté n^k fois. Au cours de ρ' aucun $\mathtt{R}_\ell$ pour $\ell \geq k$ n'est effectué, et au moins n^k incréments hiérarchisés $\mathtt{IC}_\ell$ pour $\ell \geq k$ sont effectués. Par le lemme 5.21, on obtient $coût(\rho') \geq n$, et par conséquent $LAR_\Gamma(u) \geq n$. On en déduit que :

$$coût(u) \preccurlyeq_\alpha LAR_\Gamma(u) \qquad \text{pour } \alpha(n) = k(n+1)^k - 1 \ .$$

CQFD

Il s'agit maintenant de répondre au problème original, *c.-à-d.* de transformer un B-automate en B-automate hiérarchique. L'idée est de composer cette automate à l'automate LAR, celui-ci étant chargé de traduire la condition B en condition hB. La construction est en fait une composition de transducteurs.

Formellement, étant donné un B-automate $\mathcal{A} = \langle Q, A, I, F, \Gamma, \Delta\rangle$, et soit $LAR_\Gamma = \langle P, A_\Gamma, H_k, \delta\rangle$, nous construisons l'automate $\mathcal{A}' = \langle Q \times P, A, I \times P, F \times P, H_k, \Delta'\rangle$ tel que :

$$\Delta' = \{((q,p), a, h, (q',p')) \ : \ (q,a,m,q') \in \Delta,\ (p,m,h,p') \in \delta\}\ .$$

L'automate hiérarchique $\mathcal{A}'$ est $\approx$-équivalent à $\mathcal{A}$. En fait, nous verrons des versions beaucoup plus élaborées de ce type de construction lors de la partie III et plus précisément au cours de la section 7.7.

Cette construction achève la preuve du théorème 5.19.

5.5 Équivalence avec les expressions B-rationnelles

Un autre résultat fondateur des langages réguliers est l'équivalence avec les expressions rationnelles. Là encore des résultats similaires sont possibles dans le cadre des automates à coût. Nous présentons les notions concernées dans cette section.

Une **expression B-rationnelle** (ou **B-expression**) respecte la syntaxe suivante :

$$E ::= \quad \varepsilon \quad | \quad a \quad | \quad E + E \quad | \quad E \cdot E \quad | \quad E^* \quad | \quad E^B\ .$$

La sémantique $[\![e]\!]$ d'une expression régulière e est une application de A^* dans $\mathbb{N} \cup \{\infty\}$ définie par induction en associant à chaque mot u :

$$[\![E]\!]_{\mathtt{B}} = \inf\{n \ : \ E[\leq n]\},$$

où $E[\leq n]$ représente le langage obtenu en évaluant l'expression rationnelle E en interprétant chacun des opérateurs L^B comme $L^{\leq n}$ avec

$$L^{\leq n} = \{u_1 \dots u_k \ : \ k \leq n,\ u_1, \dots, u_k \in L\}\ .$$

Exemple 5.23. Sur l'alphabet $\{a, b\}$,
- $[\![a^B]\!]_{\mathtt{B}}(u)$ est $|u|$ si $u \in a^*$, et ∞ sinon,
- $[\![b^*(ab^*)^B]\!]_{\mathtt{B}} = |u|_a$.

Théorème 5.24. *Les expressions* B*-rationnelles sont $\approx$-équivalentes avec les* B*-automates.*

Idée. Des B-automates vers les B-expressions. Sans perte de généralité, nous partons d'un B-automate hiérarchique $\mathcal{A}$. La preuve s'effectue par récurrence sur le nombre de transitions de l'automate. Le cas de base est celui d'un automate sans transition. Deux cas sont possibles. S'il existe un état à la fois initial et final, la fonction acceptée est $[\![\varepsilon]\!]$

sinon, il s'agit de $[\![\emptyset]\!]$. Supposons maintenante que l'automate contienne une transition, et considérons en particulier une (p, a, h, q) dont l'action h est maximale pour l'ordre $\mathtt{R}_0 \sqsubset \mathtt{IC}_1 \sqsubset \mathtt{R}_1 < \ldots$. Soient $\mathcal{A}'_{I,F}$, $\mathcal{A}'_{I,p}$, $\mathcal{A}'_{q,p}$ et $\mathcal{A}'_{q,F}$ les automates obtenus de $\mathcal{A}$ en enlevant la transition (p, a, h, q), et en choisissant I ou $\{q\}$ comme états initiaux, et F ou $\{p\}$ comme états finaux en concordance avec l'indice de l'automate. Par hypothèse d'induction, il existe des B-expressions correspondantes, nommément $E'_{I,F}$, $E'_{I,p}$, $E'_{q,p}$ et $E'_{q,F}$. Nous posons maintenant :

$$E = \begin{cases} E'_{I,F} + E'_{I,p} a (E'_{q,p} a)^B E'_{q,F} & \text{si } h \text{ est de la forme } \mathtt{IC}_\ell \\ E'_{I,F} + E'_{I,p} a (E'_{q,p} a)^* E'_{q,F} & \text{sinon} \end{cases}$$

Il est aisé de montrer que $[\![E]\!] \approx [\![\mathcal{A}]\!]$.

Des expressions B-rationnelles vers les B-automates. La preuve est par récurrence sur la B-expression. La construction est particulièrement simple en s'autorisant les ϵ-transitions. Le cas le plus intéressant correspond à une expression de la forme E^B. Dans ce cas, nous appliquons l'hypothèse de récurrence afin d'obtenir un automate $\mathcal{A}$ qui calcule une fonction équivalente à $[\![E]\!]$. Nous construisons alors le nouvel automate $\mathcal{A}'$ à partir de $\mathcal{A}$ comme suit :

- ajouter un état initial et final isolé,
- ajouter un nouveau compteur γ (la construction peut être améliorée en choisissant un compteur γ déjà existant, pourvu que celui-ci ne soit jamais remis à 0 dans $\mathcal{A}$),
- ajouter pour tout état initial q et tout état final p une nouvelle transition $(p, \epsilon, \mathtt{IC}_\gamma, q)$, où $\mathtt{IC}_\gamma$ met à 0 tous les compteurs, sauf γ auquel l'action `ic` est attribuée.

Dans le cas d'une expression E^* la même construction est utilisée, à la différence près qu'aucun compteur n'est ajouté, et que les transitions rajoutées remettent tous les compteurs à zéro. Là encore, il est possible de montrer que cette construction traduit une B-expression en un B-automate (en fait, hiérarchique) calculant la même fonction de coût. CQFD

5.6 Les S-automates

Nous présentons maintenant la forme duale d'automate : les S-automates. Ces automates sont toujours des automates non-déterministes à état fini, possédant des états initiaux et finaux ainsi qu'un ensemble fini de compteurs. Les actions élémentaires restent `i` (incrément), `r` (remise à zéro), et `c` (test), et les valeurs prises par les compteurs évoluent suivant les mêmes règles que pour les B-automates. Deux différences les séparent des B-automates. Tout d'abord, les actions élémentaires sont associées différemment : les actions utilisées par l'automate sont maintenant ε, `i`, `r` et `cr` (au lieu de ε, `ic` et `r`). Ensuite, l'automate calcule maintenant le maximum sur toutes les exécutions acceptantes du minimum des valeurs observées au cours de l'exécution.

Formellement, étant donné l'ensemble des compteurs Γ, et une séquence $u \in \{\varepsilon, \mathtt{i}, \mathtt{r}, \mathtt{c}\}$, $C(u)$ dénote (comme précédemment) l'ensemble des valeurs testées pendant l'exécution de

u. Nous posons,

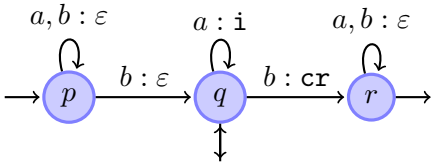
$$coût_{\mathtt{S}}(u) = \inf C(u) \ .$$

Là encore, $coût_{\mathtt{S}}$ est naturellement étendu aux exécutions de manière naturelle. Étant donné un S-automate $\mathcal{A}$, et un mot u, posons

$$[\![\mathcal{A}]\!]_{\mathtt{S}}(u) = \sup\{coût_{\mathtt{S}}(\sigma) \ : \ \sigma \text{ exécution acceptante sur } u\} \ .$$

Une dernière caractéristiques est que l'on autorise les automates à effectuer à la fin du mot une action sur les compteurs. Le formalisme des jeux introduira des notations consistantes pour ce type de phénomènes.

Exemple 5.25. Nous ne donnerons qu'un seul exemple de S-automate.



Cet automate compte la taille du plus long block de lettres a consécutives suivi d'une occurrence de b.

Sans nous attarder plus sur cet objet, précisons que, comme dans le cas des B-automates, un certain nombre de constructions sont simples à établir sur les S-automates.

Proposition 5.26. *Les fonctions calculées par les* S*-automates sont closes sous* min, max *et* sup*-projection.*

Remarquons en particulier qu'il était élémentaire d'effectuer l'inf-projection sur les B-atomates. Cette opération devient très délicate pour les S-automate, et nécessite d'être considérée modulo $\approx$. Pour les S-automates, dont la sémantique est duale, la situation est inversée. Il est très simple d'effectuer une sup-projection, alors que l'inf-projection devient délicate, et ne peut être envisagée que modulo $\approx$.

Les S-automates seront redéfinis, et utilisés dans la partie sur les arbres.

5.7 Les résultats centraux d'équivalence

Nous avons présenté jusqu'à présent les B-automates en nous concentrant sur les résultats simples à obtenir. Le point crucial est de comparer les fonctions qu'ils calculent aux autres modèles présentés dans ce document. Nous avons vu que les B-automates se traduisaient naturellement en logique monadique de coût (proposition 5.7). En fait, il en va de même pour les S-automates. Comme nous avons vu au chapitre précédent, les fonctions définissables en logique monadique de coût sur les mots sont reconnaissables par monoïdes de stabilisation. Afin de clore ces cercles d'équivalence, il ne nous manque que le résultat suivant.

Proposition 5.27. *Les fonctions de coût reconnaissables sont acceptées par* B*-automates et par* S*-automates.*

Idée. Nous ne donnons une description de cette preuve que pour les B-automates (le cas dual étant similaire). En fait, plus précisément, nous montrons comment traduire une fonction reconnaissable de coût en expression B-rationnelle. Considérons donc une fonction de coût reconnue par $\mathbf{M}, h, I$. Il s'agit de construire une expression B-rationnelle E telle que $[\![E]\!](u) \leq n$ (où de manière équivalente $u \in E[\leq n]$) si et seulement s'il existe un n-sous-calcul sur $\tilde{h}(u)$ de hauteur au plus $3|M|$ et de valeur dans $M \setminus I$. Bien entendu, une telle expression accepte $[\![\mathbf{M}, h, I]\!]^{--}_{3|M|}$, et il s'agit par définition de la fonction de coût reconnue par $\mathbf{M}, h, I$.

La construction s'effectue naturellement par récurrence sur la hauteur du sous-calcul. Il s'agit de construire, par récurrence sur l'entier ℓ et pour tout $a \in \mathbf{M}$ une expression B-rationnelle $E_{a,\ell}$ telle que pour tout mot u,

> $u \in E_{a,\ell}[\leq n]$ si et seulement s'il existe un n-sous-calcul sur $\tilde{h}(u)$ de hauteur au plus ℓ et de valeur $\geq a$.

Les règles de constructions récurrentes sont alors très naturelles. Le cas de la hauteur $\ell = 0$ (il est classique de prendre la convention que les arbres restreints aux feuilles sont de hauteur 0) est naturellement :

$$E_{a,0} = \sum_{c \in \mathbb{A},\, h(c) \geq a} c \ .$$

Pour $\ell \geq 0$, l'expression s'écrit :

$$E_{a,\ell+1} = E_{a,\ell} + \sum_{a \leq b \cdot c} E_{b,\ell} E_{b,\ell} + \sum_{a \leq e = e \cdot e} E^B_{e,\ell} + \sum_{a \leq e^\sharp,\, e = e \cdot e} E^*_{e_\ell} \ .$$

La preuve de correction s'obtient directement en déroulant les définitions. Ainsi, le premier terme correspond aux sous-calculs de hauteur au plus ℓ, le second au sous-calculs dont la racine est un nœud binaire, le troisième aux sous-calculs dans la racine est un nœud idempotent (le degré ne doit pas excéder n, et pour cette raison l'exposant est B) et enfin le dernier terme produit des sous-calculs dont la racine est soit un nœud de stabilisation soit un nœud idempotent (l'expression est incapable de distinguer ces deux cas).

L'expression B-rationnelle finale est alors simplement

$$E = \sum_{a \in M \setminus I} E_{a,3|M|} \ .$$

CQFD

À ce stade, nous avons montré que tous les formalismes vus jusqu'à présent étaient effectivement équivalents : logique monadique de coût, monoïdes de stabilisation, B-automates

(hiérarchiques ou non), S-automates et expressions B-régulières. Nous avons passé sous silence la notion d'expression S-régulière. Nous verrons aussi que les formes d'automates alternants sont également équivalent, au cours de la partie suivante, ainsi que la notion de déterminisme en histoire, elle aussi permettant d'obtenir des automates équivalents.

Si nous ne retenons de la proposition 5.27 que sa conséquence sur les automates, nous obtenons le théorème suivant :

Théorème 5.28 (dualité)**.** *Une fonction de coût est acceptée par un* B*-automate si et seulement si elle est acceptée par un* S*-automate.*

Ce théorème est celui qu'il conviendra d'étendre aux arbres. Il correspond à un résultat de complémentation. En effet, considérons l'équivalence ci-dessus pour les automates sans compteurs, *c.-à-d.* les automates non-déterministes habituels. Alors le théorème se reformule en «les automates non-déterministes de mots finis sont clos sous complément.»

5.8 Le cas des mots infinis

Le cas des arbres infinis sera développé au cours de la partie suivante. De nouvelles techniques permettant de travailler avec les automates seront alors présentées. En particulier, les notations seront modifiées et rendues plus précises.

Nous ne donnons ici que quelques idées très superficielles du cas des mots de longueur ω, et plus précisément, pourquoi, tout ce qui a été raconté dans ce chapitre se transfère sans modification au cas des mots infinis. Nous ne traitons que le cas des automates de B-Buchi, c'est à dire des automates qui sont la combinaison naturelle des B-automates présentés ci-dessus, et des automates de Büchi.

Un **automate de B-Büchi** a une définition identique à celle d'un B-automate si ce n'est que l'ensemble d'états finaux F est remplacé par un ensemble d'états de Büchi B. Une exécution (de longueur ω) se définit comme dans le cas des mots finis. Une exécution est dite **acceptante** si elle part de l'état initial et rencontre infiniment souvent un état de l'ensemble B. Le reste des définitions est identique, à commencer par la signification de $[\![\mathcal{A}]\!]_{\mathtt{B}}$, où $\mathcal{A}$ est un automate de B-Büchi.

Des **expressions ω-B-rationnelles** s'obtiennent également naturellement des expressions B-régulière en autorisant la construction supplémentaire E^ω qui itère une infinité de fois l'expression E.

Exemple 5.29. Considérons par exemple sur l'alphabet $\{a, b\}$ la fonction qui à chaque mot associe

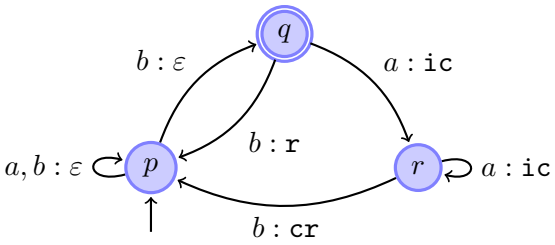
- la valeur ∞ si il n'y a qu'un nombre fini d'occurrences de b,
- sinon le plus petit n tel qu'infiniment souvent un segment maximal de a consécutifs a une longueur au plus n (il se peut que n soit aussi infini dans ce cas).

Cette fonction se décrit très simplement au moyen de la expression ω-B-rationnelle suivante :

$$E = ((a+b)^* b a^B b)^\omega \ .$$

Cette expression décompose le mot en une infinité de facteurs satisfaisants $(a+b)^* b a^B b$, c'est à dire qu'elle trouve une infinité de facteurs disjoints dans le mots de $ba^B b$ ce qui revient à trouver infiniment souvent un petit segment de a consécutifs.

Un automate de B-Büchi pour cette fonction se décrit comme suit.



La convention est ici de représenter les états de Büchi par un double cercle.

La théorie se prolonge naturellement au cas des ω-mots.

Proposition 5.30. *Les fonctions de coût sur les mots infinis les propriétés suivantes sont équivalentes*

- *être reconnaissable par algèbre de Wilke de stabilisation,*
- *être définissable en logique monadique de coût,*
- *être accepté par un ω-B-automate,*
- *être décrit par une expression ω-B-rationnelle.*

En fait, les constructions restent les mêmes.

Chapitre 6

Caractérisation de propriétés

Une belle page de la théorie des langages concerne la caractérisation (si possible effective) de familles de langages réguliers. Elle a été ouverte par Schützenberger quand il caractérisa les langages rationnels exprimables par des expressions régulières sans étoile (avec complément) comme ceux dont le monoïde syntaxique est apériodique [93]. Cette caractérisation fut complétée ultérieurement par McNaughton et Papert [72], ainsi que, dans une optique différente, par Kamp [50].

Au cours de ce bref chapitre, nous allons présenter quelques résultats de caractérisation comparables concernant les fonctions régulières de coût. Nous commençons par introduire une notion de monoïde de stabilisation syntaxique (section 6.1) qui est un outil important pour les résultats de caractérisations. La section 6.2 traite de la classe temporelle, c'est à dire la classe des fonctions de coût dont la sémantique a été restreinte pour ne mesurer que les intervalles. La section 6.3 traite de la classe apériodique, qui est le pendant du résultat de Schützenberger-McNaughton-Papert-Kamp pour les fonctions de coût. La section 6.4 conclut en présentant quelques questions ouvertes.

Ces résultats sont dus à Denis Kuperberg et sont détaillés dans son document de thèse [59].

6.1 Le monoïde de stabilisation syntaxique

Un prérequis pour bon nombre de résultats de caractérisations est de disposer d'un bon objet à analyser. Souvent, dans le cas des langages, il s'agit d'analyser le monoïde ou le semigroupe syntaxique du langage (ce n'est pas toujours le cas, et certains résultats reposent sur d'autres descriptions des langages). Cet objet est le pendant de l'automate déterministe minimal, dans le cadre des monoïdes. Il s'agit du plus petit monoïde (ou semigroupe) qui permette de reconnaitre le langage. Nous suivons la même approche dans le cadre des fonctions régulières de coût sur les mots finis, et nous introduisons une notion de monoïde de stabilisation syntaxique. Précisons bien «une» car il existe plusieurs manières de donner

une telle définition, qui ne sont pas toutes équivalentes en général, mais le sont dans le cas des fonctions régulières de coût. La situation est similaire à celle des algèbres de Wilke.

Quelques définitions nous seront utiles pour définir le monoïde de stabilisation syntaxique d'une fonction. Un **♯-contexte** C est une ♯-expression avec un «trou». Nous noterons $C[E]$, où E est une ♯-expression la ♯-expression obtenue de C en insérant E dans le trou. Considérons une fonction de coût f sur l'alphabet $\mathbb{A}$, et une ♯-expression E sur $\mathbb{A}$. On dira que f est **bornée sur** E s'il existe un entier $k > 0$ tel que f est bornée sur Dépliages$^+(E, k)$. Remarquons que si f est bornée sur Dépliages$^+(E, k)$, alors elle l'est aussi sur Dépliages$^+(E, k')$ pour tout k' multiple de k. Une manière simple détablir ce fait est de simplement remarquer que Dépliages$^+(E, k') \subseteq$ Dépliages$^+(E, k)$. Cet argument d'inclusion sera largement réutilisé.

Étant donné une fonction de coût f sur l'alphabet $\mathbb{A}$, définissons la relation de préordre $\leq_f$ sur l'ensemble des ♯-expressions sur $\mathbb{A}$ par :

$$E \leq_f F \qquad \begin{array}{l}\text{si pour tout ♯-contexte } C,\\ f \text{ bornée sur } C[E] \text{ implique } f \text{ bornée sur } C[F].\end{array}$$

Le préordre $\leq_f$ est appelé la **congruence syntaxique de** f. La relation d'équivalence correspondante, $\leq_f \cap \geq_f$, est notée $\equiv_f$.

Lemme 6.1. *Si f est reconnue par $\mathbf{M}, h, I$ où $\mathbf{M} = (M, \cdot, \leq, \sharp)$, alors pour toutes ♯-expressions E, F sur $\mathbb{A}$,*

$$\mathit{valeur}(h(E)) \leq \mathit{valeur}(h(F)) \quad \mathit{implique} \quad E \leq_f F \ .$$

Démonstration. Considérons un ♯-contexte C. Supposons que f est bornée sur $C[E]$, *c.-à-d.* qu'il existe k tel que f est bornée sur Dépliages$^+(C[E], k)$. Cela reste en particulier vrai pour un k multiple de $|M|!$. La proposition 4.25 nous enseigne alors que $\mathit{valeur}(h(C[E])) \notin I$. En utilisant l'hypothèse que $\mathit{valeur}(h(E)) \leq \mathit{valeur}(h(F))$, on obtient $\mathit{valeur}(h(C[E])) \leq \mathit{valeur}(h(C[F]))$ et donc $\mathit{valeur}(h(C[F])) \notin I$. Par la proposition 4.25, il s'ensuit que f est bornée sur Dépliages$^+(C[F], k)$, *c.-à-d.* que f est bornée sur F. CQFD

Corollaire 6.2. *Si f est reconnaissable par monoïde de stabilisation*[1]*, $\equiv_f$ est d'index fini.*

Le lemme suivant montre que les ♯-expressions quotientées par $\equiv_f$, équipées de l'ordre $\leq_f$, possèdent toutes les propriétés d'un monoïde de stabilisation (si ce n'est celle d'être fini, en général).

Lemme 6.3. *Pour toutes ♯-expressions E, F, G, E', F' et toute fonction de coût f,*

1. *si $E \leq_f F$, alors $D[E] \leq_f D[F]$ pour tout ♯-contexte D.*
2. *si $E \leq_f E'$ et $F \leq_f F'$, alors $EF \leq_f E'F'$,*

1. Rappelons que par définition, les monoïdes de stabilisation sont finis.

3. *si* $E \leq_f F$, $E^{\omega\sharp} \leq_f F^{\omega\sharp}$,
4. $(EF)G \equiv_f E(FG)$
5. $(EF)^{\omega\sharp}E \equiv_f E(FE)^{\omega\sharp}$,
6. *si* $EE \equiv_f E$ *(on dira que* E *est idempotent),* $E^{\omega\sharp} \leq_f E$,
7. *si* E *est idempotent,* $E^{\omega\sharp}E \equiv_f E^{\omega\sharp}$,
8. *si* E *est idempotent,* $(E^{\omega\sharp})^{\omega\sharp} \equiv_f E^{\omega\sharp}$.

Démonstration. Dans la suite, C désigne un $\sharp$-contexte arbitraire.

1. Supposons que $E \leq_f F$. Si f est bornée sur $C[D[E]]$, alors f est bornée sur $C[D[F]]$, simplement en considérant $C[D[*]]$ comme un $\sharp$-contexte. Ainsi, $D[E] \leq_f D[F]$.

2. En utilisant deux fois le cas précédent, $EF \leq_f E'F \leq_f E'F'$.

3. Conséquence directe du premier item.

4. Pour tout k, Dépliages$^+(C[E(FG)], k)$ = Dépliages$^+(C[(EF)G], k)$. Ainsi, si f est bornée sur l'un de ces ensembles, elle l'est aussi sur l'autre.

5. Idem : Dépliages$^+(C[(EF)^{\omega\sharp}E], k)$ = Dépliages$^+(C[E(FE)^{\omega\sharp}], k)$.

6. Supposons $EE \equiv_f E$, et supposons que f est bornée sur $C[E^{\omega\sharp}]$. Cela signifie qu'il existe un entier k tel que f est bornée sur Dépliages$^+(C[E^{\omega\sharp}], k)$. Or Dépliages$^+(C[E^{\omega\sharp}], k) \subseteq$ Dépliages$^+(C[E^k], k)$. Il s'ensuit que f est bornée sur $C[E^k]$, et donc sur $C[E]$ car $E^k \equiv E$ (obtenu en appliquant $k-1$ fois le premier item sur$EE \equiv_f E$).

7. Similairement, si E est idempotent, supposons pour commencer que f est bornée sur $C[E^{\omega\sharp}]$. Cela signifie qu'il existe un entier k tel que f est bornée sur Dépliages$^+(C[E^{\omega\sharp}], k)$. Comme Dépliages$^+(C[E^kE^{\omega\sharp}], k) \subseteq$ Dépliages$^+(C[E^{\omega\sharp}], k)$, on obtient que f est bornée sur $C[E^kE^{\omega\sharp}]$, et donc sur $C[EE^{\omega\sharp}]$ (car $E^k \equiv_f E$, cf. preuve du cas précédent). Pour l'autre direction, si f est bornée sur $C[EE^{\omega\sharp}]$, il existe un entier k tel que f est bornée sur Dépliages$^+(C[EE^{\omega\sharp}], k)$. Il s'ensuit que f est bornée sur $C[E^kE^{\omega\sharp}]$. Or Dépliages$^+(C[E^{\omega\sharp}], 2k) \subseteq$ Dépliages$^+(C[E^kE^{\omega\sharp}], k)$. Donc f est bornée sur $C[E^{\omega\sharp}]$. Ainsi $EE^{\omega\sharp} \equiv_f E^{\omega\sharp}$.

8. Il suffit de montrer que $E^{\omega\sharp} \leq_f (E^{\omega\sharp})^{\omega\sharp}$. Supposons que f est bornée sur $E^{\omega\sharp}$. Il existe k tel que f est bornée sur Dépliages$^+(E^{\omega\sharp}, k)$. Or Dépliages$^+((E^{\omega\sharp})^{\omega\sharp}, k) \subseteq$ Dépliages$^+(E^{\omega\sharp}, k)$. Donc f est bornée sur $(E^{\omega\sharp})^{\omega\sharp}$.

CQFD

Étant donnée une fonction de coût sur l'alphabet $\mathbb{A}$, définissons son **semigroupe de stabilisation syntaxique** par :

- Les éléments sont les classes d'équivalences de $\sharp$-expressions pour $\equiv_f$.
- L'ordre est $\leq_f$.
- Comme $\leq_f$ est une congruence, les opérations de produit et de stabilisation sont définies de manière canonique.

Si l'on autorise l'utilisation de ε dans les $\sharp$-expressions, alors l'objet ainsi construit est le **monoïde de stabilisation syntaxique**. Le semigroupe de stabilisation syntaxique de f est dénoté $\mathbf{S}(f)$, et le monoïde de stabilisation syntaxique de f, $\mathbf{M}(f)$.

D'après le lemme 6.3, toutes les propriétés requises par un semigroupe de stabilisation (*resp.* un monoïde de stabilisation) sont satisfaites par le «semigroupe de stabilisation syntaxique» (*resp.* par le monoïde de stabilisation syntaxique), si ce n'est la propriété d'être fini. Le lemme 6.1 énonce que si la fonction f est reconnaissable, alors le semigroupe/monoïde de stabilisation syntaxique est lui même fini.

Lemme 6.4. *Si f est reconnue par $\mathbf{M}, h, I$, alors :*
- *$\mathbf{M}(f)$ est un quotient d'un sous-monoïde de stabilisation de $\mathbf{M}$,*
- *$\mathbf{M}(f), h, I$ reconnaît f, où h envoie chaque lettre a sur la $\sharp$-expression a, et I contient les $\sharp$-expressions E telles que f n'est pas bornée sur E.*

Nous n'établissons pas cet énoncé très classique.

Lemme 6.5. *Étant donnée une fonction de coût reconnaissable par monoïde de stabilisation, son monoïde de stabilisation syntaxique est calculable.*

Là encore, il s'agit de techniques très classiques. Il s'agit de ne garder que les éléments accessibles à partir des lettres par produit et stabilisation, puis de quotienter par la congruence la plus grossière qui soit compatible avec l'idéal. Cette procédure est polynomiale.

6.2 Fragment temporel

Le premier résultat de caractérisation connu pour les fonction régulières de coût concerne la classe temporelle. Cette notion n'a pas d'équivalents dans le cas des langages. Elle consiste à restreindre la possibilité de mesurer le nombre d'occurrences d'évènements à des séquence d'évènements consécutifs. Nous allons donner plusieurs caractérisations équivalentes de cette classe, dont l'une est effective. Ces travaux sont le résultat d'une collaboration avec Denis Kuperberg et Sylvain Lombardy [30].

La première définition de cette classe est une variante de la logique monadique de coût, qui s'applique aux mots. La **logique monadique de coût temporelle** s'obtient en autorisant dans la logique monadique de coût la possibilité d'utiliser positivement le prédicat $|y - x| \leq N$ en lieu et place de $|X| \leq N$. La signification de ce prédicat est de tester que la distance entre x et y est au plus N. Il s'agit d'un fragment de la logique monadique de coût puisque $|y - x| \leq N$ revient à tester que l'intervalle minimal contenant x et y est de longueur au plus $N + 1$, et cet intervalle se définit en logique monadique.

Exemple 6.6. La formule suivante énonce que tous les intervalles consécutifs de a ont une longueur d'au plus $N - 1$:

$$\forall x \forall y \ \ ((x \leq y) \wedge \forall z \ x \leq z \leq y \to a(z)) \to |y - x| \leq N \ .$$

C'est à dire que cette formule définit la fonction qui à $a^{n_0}ba^{n_1} \dots ba^{n_k}$ associe $\max(0, n_0 - 1, \dots, n_k - 1)$ (où, à $\approx$ près, $\max(n_0, \dots, n_k)$).

Nous verrons que toutes les fonctions de coût sur les mots finis ne peuvent être décrites par ce moyen. De tels résultats, même s'il peuvent être établis directement, seront des conséquences très directes des résultats de caractérisation qui seront présentés au cours de cette section. En particulier, la fonction de coût qui à $u \in \{a, b\}^*$ associe le nombre d'occurrences de la lettre a ne peut être décrite par ce fragment logique. Informellement, la capacité de mesurer la taille d'intervalles ne semble pas aider à compter le nombre d'occurrences de la lettre a, sachant que ces lettres peuvent se trouver à des distances extrêmement grandes les unes des autres. Nous verrons ce cas au cours de l'exemple 6.7.

Notre deuxième définition de la classe temporelle est algébrique. Il s'agit de restreindre les semigroupes. Remarquons ici l'usage du semigroupe plutôt que du monoïde. Cela provient du fait que pour cette classe, le mot vide ne joue pas un rôle similaire aux lettres.

Nous dirons qu'un idempotent e d'un semigroupe de stabilisation est **stable** si $e^\sharp = e$. Intuitivement, un élément est stable si le semigroupe ne compte pas le nombre d'itérations de cet élément. En effet, itérer un petit nombre de e, ce qui produit la valeur e, est indistinguable d'un grand nombre d'itérations de cet élément, ce qui produit $e^\sharp$. Un semigroupe de stabilisation est **temporel** si :

pour tous idempotents e et $e \cdot x$, si e est stable, $e \cdot x$ l'est aussi.

Informellement, cela signifie que si l'on s'intéresse à compter le nombre d'occurrences de $e \cdot x$, alors il est obligatoire de compter le nombre de e.

Exemple 6.7. L'exemple 4.6 correspond à la fonction qui calcule la plus grande longueur d'un segment de a consécutifs. Pour voir cet exemple comme un semigroupe, il convient de se restreindre à l'ensemble des éléments qui sont engendrés par l'image des lettres, *c.-à-d.* dans notre cas $\{a, b\}$. En particulier, cela élimine l'élément neutre 1. Il nous reste le semigroupe suivant.

	a	b	0	$\sharp$
a	a	b	0	0
b	b	b	0	b
0	0	0	0	0

Ce semigroupe de stabilisation est temporel. En effet, a est le seul idempotent instable, et il ne s'écrit ni $b \cdot x$ ni $0 \cdot x$ pour aucun x.

Considérons maintenant l'exemple 4.4 qui compte le nombre d'occurrences de la lettre a, et voyons-le comme un semigroupe. Cette fois-ci, se restreindre aux éléments engendrés par les lettres $\{a, b\}$ n'élimine aucun élément. Le semigroupe est donc défini par la table suivante.

	b	a	0	$\sharp$
b	b	a	0	b
a	a	a	0	0
0	0	0	0	0

Ce semigroupe de stabilisation n'est pas temporel car b est stable, alors que $a = a \cdot b$ ne l'est pas. Cela s'interprète comme : ce monoïde compte les occurrences de la lettre a, mais ne s'intéresse pas aux itérations de b qui peuvent les séparer.

Notre dernière caractérisation est en termes d'automates. Dans ce cas, forcer le modèle à ne compter que des évènements consécutifs est très simple. Il suffit d'interdire l'action ε. Cela donne la définition suivantes. Un `B`-automate est **temporel** s'il n'utilise que les actions `ic` et `r`. Un `S`-automate est **temporel** s'il n'utilise que les actions `i` et `cr`. Les `B`-automates temporels à un compteur correspondent exactement aux **automates de désert** introduits indépendamment par Bala et Kirsten [1, 51].

Toutes ces caractérisations sont réunies au sein du théorème suivant.

Théorème 6.8 ([30]). *Pour une fonction reconnaissable de coût sur le mots, les propriétés suivantes sont équivalentes :*

- *être définissable en logique monadique de coût temporelle,*
- *posséder un semigroupe syntaxique temporel,*
- *être reconnu par un semigroupe temporel,*
- *être reconnu par un* `B`*-automate temporel,*
- *être reconnu par un* `B`*-automate temporel à un compteur,*
- *être reconnu par un* `S`*-automate temporel,*
- *être reconnu par un* `S`*-automate temporel à un compteur.*

De plus, ces équivalences sont effectives et la classe est décidable.

En fait, les techniques développées dans [30] sont plus informatives, et permettent de prouver la décidabilité de la domination entre formules de la logique monadique temporelle sans passer par la théorie complète des fonctions régulières de coût. Remarquons finalement que ce fragment des fonctions régulières de coût contient toutes les fonctions caractéristiques des langages réguliers. En ce sens, cette classe n'a aucun équivalent dans la théorie des langages réguliers.

6.3 Fragment apériodique

Comme nous l'avons mentionné au cours de l'introduction de ce chapitre, le résultat de caractérisation du fragment apériodique des langages réguliers est le résultat qui a initié l'étude de classes particulières de langages réguliers. Dans cette section nous montrons les résulats connus concernant la classe apériodique correspondante pour les fonctions régulières de coût. Ces travaux ont été menés par Denis Kuperberg au cours de sa thèse [58, 59].

Rappelons tout d'abord le fameux résultat de Schützenberger-McNaugthon-Papert, que nous avons étendu par le résultat de Kamp qui caractérise la classe au moyen de la logique temporelle.

Théorème 6.9 ([93, 72, 50]). *Pour un langage régulier de mots finis, les propriétés suivantes sont équivalentes :*

- *être définissable en logique du premier ordre,*
- *être définissable par une expression rationnelle sans étoile (formée à partir des langages finis en utilisant la concaténation, l'union, et la complémentation),*
- *est reconnaissable par un monoïde* **apériodique***, c.-à-d. que pour tout élément x, il existe un entier n tel que $x^{n+1} = x$,*
- *avoir un monoïde syntaxique apériodique,*
- *être reconnaissable par un monoïde* **groupe-trivial***, c.-à-d. dont tous les sous-monoïdes qui sont des groupes sont triviaux,*
- *avoir un monoïde syntaxique groupe-trivial,*
- *être définissable en logique temporelle* LTL,
- être reconnu par un automate sans compteur [2],
- avoir un automate déterministe minimal sans compteur.

Dans cette section, nous présentons les résultats connus concernant la classe de fonctions régulières de coût correspondante. Bien entendu, les définitions de monoïdes apériodiques ou groupe-trivial restent identique. Un monoïde de stabilisation est **apériodique** (*resp.* **groupe-trivial**) si le monoïde sous-jacent est **apériodique** (*resp.* **groupe-trivial**).

Nous commençons par définir la **logique du premier-ordre de coût**. Elle se définit comme la logique du premier ordre, mais augmentée de la nouvelle construction

$$\forall^{\leq N} x\ \varphi$$

qui doit apparaître positivement dans la formule. La construction $\forall^{\leq N} x\ \varphi$ signifie que φ est vérifiée pour tout x, sauf pour au plus N. Dit différemment, cela signifie que, sur la structure $\mathcal{S}$ d'univers $\mathcal{U}$, et étant donné des valuations $\bar{v}$ des autres variables libres,

$$\mathcal{S} \models \forall^{\leq N} x\ \varphi(x, \bar{v}) \quad \text{si} \quad \{u \in \mathcal{U}\ :\ \mathcal{S} \models \varphi(u, \bar{v})\} \leq N\ .$$

Il s'agit bien d'un fragment de la logique monadique de coût puisque $\forall^{\leq N} x\ \varphi(x)$ est équivalent à la formule

$$\exists Y\ |Y| \leq N \wedge \forall x\ x \in Y \vee \varphi(x)\ .$$

La dernière caractérisation de cette classe que nous donnerons est en terme de logique temporelle. La logique temporelle **LTL de coût** possède la syntaxe suivante :

$$\varphi ::= a \quad | \quad \varphi \wedge \varphi \quad | \quad \neg\varphi \quad | \quad \mathtt{X}\varphi \quad | \quad \varphi \mathtt{U} \varphi \quad | \quad \varphi \mathtt{U}^{\leq N} \varphi,$$

où la construction $\varphi \mathtt{U}^{\leq N} \varphi$ doit obligatoirement apparaître positivement dans la formule. La signification de la plupart des construction de cette définition est classique. La nouvelle construction $\varphi \mathtt{U}^{\leq N} \psi$ est informellement d'énoncer que «ψ est vrai dans le futur, et que

2. Un «compteur» est ici un mot u et deux états $p \neq q$ tels qu'il existe un chemin étiqueté par u qui va de p à q et un autre étiqueté par u^k $(k > 0)$ de q à p.

dans l'intervalle, φ est toujours vrai, sauf pour au plus n instants». Nous donnons ici la sémantique de manière concise en traduisant toute formule φ de la logique LTL de coût en une formule de logique du premier-ordre de coût $\varphi^*(x)$ (celle-ci possède une unique variable libre x). Les règles de traduction sont les suivantes :

$$\begin{aligned}
a^*(x) &\stackrel{\text{def}}{=} a(x) \\
(\varphi \wedge \psi)^*(x) &\stackrel{\text{def}}{=} \varphi^*(x) \wedge \psi^*(x) \\
(\neg\varphi)^*(x) &\stackrel{\text{def}}{=} \neg\varphi^*(x) \\
(\mathtt{X}\varphi)^*(x) &\stackrel{\text{def}}{=} \exists y\ y = x+1 \wedge \varphi^*(y) \\
(\varphi \mathtt{U} \psi)^*(x) &\stackrel{\text{def}}{=} \exists y\ y \geq x \wedge \psi^*(y) \wedge \forall z\ x \leq z < y\ \rightarrow\ \varphi^*(y) \\
(\varphi \mathtt{U}^{\leq N} \psi)^*(x) &\stackrel{\text{def}}{=} \exists y\ y \geq x \wedge \psi^*(y) \wedge \forall^{\leq N} z\ x \leq z < y\ \rightarrow\ \varphi^*(y)
\end{aligned}$$

Là encore, tous ces formalismes coïncident.

Théorème 6.10 ([58, 59])**.** *Les propriétés suivantes sont équivalentes pour une fonction régulière de coût sur les mots finis :*

- *être définissable en logique du premier-ordre de coût,*
- *être reconnu par un monoïde de stabilisation apériodique, et en particulier, avoir un monoïde de stabilisation syntaxique apériodique,*
- *être reconnu par un monoïde de stabilisation groupe-trivial, et en particulier, avoir un monoïde de stabilisation syntaxique groupe trivial,*
- *être définissable en logique temporelle* $\mathrm{LTL}^{\leq}$.

La contribution de [58] ne s'arrête pas à énoncer ces résultats de caractérisations. Une traduction simplement exponentielle est donnée par Kuperberg, et Kuperberg et Vanden Boom ont étendu ces résultats au cas des ω-mots [61]. Enfin, Kuperberg donne également une caractérisation de l'intersection des classes apériodiques et temporelles [59].

6.4 Problèmes ouverts

Il existe plusieurs questions ouvertes de caractérisation de classes de fonctions de coût régulières sur les mots finis. Citons ci-dessous les principales.

Bien entendu, les automates de coût possédant des compteurs, il est naturel de vouloir en caractériser le nombre.

Question ouverte 6.11. *Est-il possible de décider, étant donnée une fonction régulière de coût, si elle est reconnue par un* B*-automate non-déterministe à k compteurs (respectivement un* S*-automate) ?*

Ce que nous savons, c'est qu'être accepté par un B-automate à k compteurs est équivalent à être accepté par un hB-automate à k compteurs (conséquence du théorème 5.19). Nous savons également qu'être accepté par un B-automate sans compteur est équivalent à être accepté par un S-automate sans compteur. À part ce fait, il n'y a aucune corrélation entre les deux paramètres. En effet, il est possible de construire, pour tout $i, j \geq 1$, une fonction régulière de coût acceptée par un B-automate à i compteurs et un S-automate à j compteurs, mais par aucun B-automate à $i-1$ compteurs ni aucun S-automate à $j-1$ compteurs.

La deuxième question anticipe sur la suite du document, car elle fait référence à la classe particulièrement importante des automates déterministes en histoire. Cette notion se décline en B-automates et en S-automates. Ces automates acceptent, eux aussi, exactement la classe des fonctions régulières de coût.

Question ouverte 6.12. *Est-il possible de décider, étant donnée une fonction régulière de coût, si elle est reconnue par un* B*-automate déterministe en histoire à k compteurs (respectivement un* S*-automate déterministe en histoire) ?*

L'une des raisons pour lesquelles ce paramètre semble plus important est mis en évidence par la conjecture suivante.

Conjecture 6.13. *Une fonction régulière de coût est acceptée par un* B*-automate déterministe en histoire à k-compteurs, si et seulement si elle est acceptée par un* S*-automate déterministe en histoire à k-compteurs.*

Pour interpréter cette énoncé, il convient de se remémorer la situation des langages réguliers sur les ω-mots, un automate déterministe utilisant une condition de Rabin à k-paire de Rabin, quand il est complémenté, se transforme en un automate déterministe utilisant une condition de Streett à k-paire, et vis-versa, simplement en complémentant la condition d'acceptation. Ce genre de conversions «syntaxiques» est spécifique des automates déterministes. La situation est différente pour les automates non-déterministes, bien que cela n'apparaisse pas de manière très évidente pour les ω-mots, pour lesquels la condition d'acceptation peut être radicalement bouleversée par une complémentation. L'opération de dualité correspondant à la complémentation dans les langages, la conjecture 6.13 énonce que cette propriété de complémenatation simple reste valable dans le cas des fonctions de coût.

Troisième partie

Les fonctions régulières de coût sur les arbres

Chapitre 7

Jeux et déterminisme en histoire

Dans ce chapitre, nous nous intéressons aux jeux correspondants aux conditions de gain des automates du chapitre précédent. D'un certain point de vue, ce chapitre peut être vu comme une préparation technique avant d'aborder le cas des arbres. Ce n'est que partiellement vrai. En effet, tout comme dans le cas de la théorie classique, l'étude des jeux est pertinente à part entière. En particulier, il est habituel de réduire un problème de model-checking symbolique à un jeu qu'il s'agit de résoudre. Bien que l'aspect de vérification de modèle avec des fonctions de coût régulières n'a pas encore été abordé, il est d'ores et déjà clair que ces questions seront résolues par des analyses fines des jeux.

Dans ce chapitre, nous présentons également la notion de *déterminisme en histoire*, c'est-à-dire une forme affaiblie (sémantique) de déterminisme qui permet de composer les automates avec les jeux. Cette notion joue également un rôle primordial dans l'étude du cas des arbres.

Beaucoup des travaux de ce chapitre ont été développés en collaboration avec Christof Löding [31, 33], parfois avec Achim Blumensath (non publié). Enfin, certains résultats ont été obtenus par Michael Vanden Boom dans le cadre de son étude de la logique monadique faible [12] qui sera présenté au chapitre 9.

Le chapitre se décompose comme suit. Dans la section 7.1 nous introduisons les jeux booléens «classiques» et nous listons les conditions de gain booléennes qui nous intéresseront à la section 7.2. La section 7.3 présente le cadre général des «jeux de coûts», et la section 7.4 énumère les conditions quantitatives que nous rencontrerons. À la section 7.5 nous présenterons tous les résultats connus (à ce jour) concernant la structure des stratégies. Enfin, nous verrons à la section 7.6 une nouvelle formalisation des automates de mots (alternants cette fois) et nous développerons la notion de déterminisme en histoire à la section 7.7. La section 7.8 utilisera cette notion pour composer les automates entre eux et avec les jeux. Enfin, la section 7.9 présentera les jeux de domination, qui jouent un rôle légèrement différent dans la théorie, mais qui nous seront utiles par la suite.

7.1 Les jeux booléens

Dans le cadre des jeux, nous utilisons les mots pour décrire les parties, qui peuvent être finies ou infinies. Dans la théorie classique, la méthode habituelle consiste à transformer les jeux afin que toutes les parties deviennent infinies. Cette astuce permet de faire disparaître élégamment un certain nombre de détails techniques. Dans notre cadre, nous ne savons malheureusement pas résoudre aussi bien les jeux dans le cas infini que dans le cas fini. Ceci signifie que notre formalisme doit rendre fidèlement compte des spécificités des parties finies, sans faire usage de cette astuce. La spécificité du cas des parties finies est qu'elle peuvent se terminer, et qu'il peut être nécessaire de connaître certaines informations supplémentaires pour déterminer le vainqueur, comme en particulier la position finale du jeu. Nous capturons ce manque d'information en ajoutant en fin de mot un symbole particulier, appelé terminateur.

Ainsi, nous appellerons dorénavant un **alphabet avec terminateurs** $\mathbb{A} = (\mathbb{A}_1, \mathbb{A}_0)$ un alphabet consistant en deux sortes : les lettres dans $\mathbb{A}_1$, d'arité 1, se comportant comme les lettres d'un alphabet usuel, et les **terminateurs** dans $\mathbb{A}_0$, d'arité 0, qui ne peuvent être utilisés qu'en fin de mot. Un **mot avec terminateur** sur $\mathbb{A}$ est une séquence dans $\mathbb{A}_1^*\mathbb{A}_0 \cup \mathbb{A}_1^\omega$, c'est à dire, soit une séquence finie de lettres d'arité 1 se terminant par un terminateur, soit un séquence infinie de lettres d'arité 1. L'ensemble des mots avec terminateur est noté $\mathbb{A}^\infty$. Par opposition, un mot de $\mathbb{A}_1^*$ est appelé un **mot partiel**. En particulier, tout alphabet, au sens habituel, peut être vu comme un alphabet avec terminateur, dans lequel il existe un unique terminateur, $\square$. Nous utilisons cette notion de mot avec terminateur dans la suite de ce document. Nous utiliserons l'alphabet particulier $\mathbb{I}$ qui ne possède qu'une seule lettre «$-$», d'arité 1, et aucun terminateur. Sur cet alphabet, il n'est possible de former qu'un mot seul mot, «$----\ldots$»

Dorénavant, $\mathcal{B}^+(X)$ dénote l'ensemble des combinaisons booléennes positives non-triviales (*c.-à-d.*, différentes de «vrai» ou «faux»), possiblement infinies (si X est infini), construites sur les éléments de X. Étant données une formule $\varphi \in \mathcal{B}^+(X)$ et une application h de X dans $\mathcal{B}^+(Y)$, nous noterons par

$$\varphi[x \leftarrow h(x)]$$

la formule φ dans laquelle chaque occurrence d'une variable x dans X est syntaxiquement remplacée par $h(x)$. Nous dénoterons également par $\overline{\varphi}$ le **dual** de φ, *c.-à-d.* la formule φ dans laquelle disjonctions et conjonctions sont inversées. Pour $A \subseteq X$ non vide, $\bigvee A$ dénote la disjonction de tous les éléments de A, et $\bigwedge A$ la conjonction de tous les éléments de A. Finalement, étant données $\varphi, \varphi' \in \mathcal{B}^+(X)$, nous écrirons $\varphi \Rightarrow \varphi'$ pour signifier que φ' est une conséquence de φ au sens habituel. Ainsi remarquons que pour A, B non vides, $\bigwedge A \Rightarrow \bigwedge B$ si et seulement si $B \subseteq A$. De même $\bigvee A \Rightarrow \bigvee B$ si et seulement si $A \subseteq B$. Similairement, si $\bigwedge A \Rightarrow \bigvee B$, cela signifie que $A \cap B \neq \emptyset$. (Notons enfin que $\bigvee A \Rightarrow \bigwedge B$ signifie que $A = B$ est un singleton. Ce dernier cas n'aura pas l'occasion d'apparaître.)

Un **jeu** (booléen) $\mathcal{G} = (V, v_0, M, \delta, W)$ est constitué :

- d'un ensemble V de **positions**,
- d'une **position initiale** $v_0 \in V$,
- d'un ensemble de **coups** $M = M_0 \uplus M_1$ où

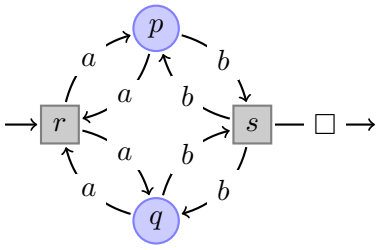
$$\begin{aligned} & M_0 \subseteq V \times \mathbb{C}_0 \times \{\bot\} \\ \text{et} \quad & M_1 \subseteq V \times \mathbb{C}_1 \times V \ , \end{aligned}$$

  où $\mathbb{C}$ est un alphabet avec terminateur, et $\bot$ est un symbole n'appartenant pas à V (l'utilisation de ce symbole ne sert qu'à traiter le cas de M_0 et M_1 de manière uniforme). Nous noterons $M(v)$ l'ensemble des coups issus de v pour $v \in V$, *c.-à-d.* l'ensemble $\{(v, a, w) : (v, a, w \in M)\}$. Les coups dans M_1 sont dits **non finaux** par opposition à ceux dans M_0, dits **finaux**.
- d'une **application de contrôle** δ qui à $v \in V$ associe $\delta(v) \in \mathcal{B}^+(M(v))$. (remarquons que par hypothèse, $\mathcal{B}^+$ ne contient que des combinaisons booléennes non triviales, et par conséquence $M(v)$ est non vide pour toute position v)
- d'une **condition de gain** W qui est un ensemble de mots avec terminateurs sur l'alphabet $\mathbb{C}$.

Un jeu sans condition de gain est appelé une **arène**. Notre définition fait la distinction entre coups finaux et non-finaux. Ce détail technique est nécessaire pour pouvoir développer les notions dans toute leur généralité.

Un jeu sera dit **chronologique** s'il est possible de numéroter les sommets par des entiers tels que tout coups du jeu augmente strictement le numéro.

Exemple 7.1. Les arènes sont représentées comme des graphes :



Dans la définition usuelle d'une arène $\delta(v)$ est soit la disjonction $\bigvee M(v)$ soit la conjonction $\bigwedge M(v)$. Dans le premier cas, la position est dite appartenir au joueur existentiel (Ève) et est présentée par un cercle (○), et dans le second au joueur universel (Adam) et est présentée par un carré (□). Dans l'exemple ci-dessus, par exemple, $\delta(r) = (r, a, p) \wedge (r, a, q)$, et $\delta(p) = (p, a, r) \vee (p, b, s)$. La position initiale ($r$ dans l'exemple ci-dessus) est représentée par une flèche entrante. Les flèches sortantes représente les coups finaux (étiquetés par des terminateurs). Dans l'exemple, $(s, \square, \bot)$ est un coup final. Ainsi, $\delta(s) = (s, b, p) \wedge (s, b, q) \wedge (s, \square, \bot)$.

Il ne manque à l'arène ci-dessus q'une condition de gain pour en faire un jeu. Par exemple, $W = (ab^+)^\omega + (ab^+)^*\square$.

Le jeu **dual** $\overline{\mathcal{G}}$ d'un jeu $\mathcal{G}$ s'obtient en remplaçant la condition de gain par son complémentaire, ainsi que l'application de contrôle par son dual (ce qui revient à échanger les positions d'Ève et les positions d'Adam dans le cas classique). Le dual d'un jeu correspond à échanger le rôle des deux joueurs. En particulier, nous donnerons dans la suite toutes les définitions pour le joueur Ève, mais en gardant à l'esprit que la notion correspondante pour Adam s'obtient simplement par dualité.

De plus, nous utilisons dans la suite $M = (M_1, M_0)$ comme un alphabet avec terminateurs. La relation M_1 équipe les positions d'une structure de graphe étiqueté. C'est en faisant référence à ce graphe que nous parlerons de cycle, de chemin, etc...

Une **partie** (sur $\mathcal{G}$) est un mot avec terminateur sur M qui forme un chemin d'origine v_0 dans le jeu. La **sortie** $Out(\pi)$ d'une partie π est le mot avec terminateur sur $\mathbb{C}$ obtenu en projetant π sur sa seconde composante. Une partie est **gagnante** pour Ève si sa sortie appartient à W. Sinon, elle est gagnante pour Adam. Un préfixe strict d'une partie est appelé **partie partielle**, et sa sortie est définie de manière similaire.

Il s'agit maintenant de décrire comment se joue un jeu. Une **stratégie pour Ève** σ_{E} est un ensemble de parties (sur $\mathcal{G}$) tel que :

- L'ensemble σ_{E} est non-vide et topologiquement **clos**, c'est à dire que pour tout mot infini $\pi = \pi_1\pi_2\dots$ tel que pour tout i, $\pi_1\dots\pi_i \in \mathit{pref}(\sigma_{\mathrm{E}})$, alors π appartient à σ_{E}, où $\mathit{pref}(\sigma_{\mathrm{E}})$ est l'ensemble des préfixes d'une partie dans σ_{E}.
- Pour toute partie partielle $\pi \in \mathit{pref}(\sigma_{\mathrm{E}})$ se terminant en v,

$$\bigwedge \sigma_{\mathrm{E}}[\pi] \quad \Rightarrow \quad \delta(v)\ , \tag{7.1}$$

  où $\sigma_{\mathrm{E}}[\pi]$ est l'ensemble $\{m \in M(v) \ : \ \pi m \in \mathit{pref}(\sigma_{\mathrm{E}})\}$, c'est à dire l'ensemble des coups envisagés par σ_{E} après la partie partielle π.

Remarque 7.2. Cette définition correspond au cas classique, ou plus précisément à la notion de quasi-stratégie. Regardons plus précisément la signification de l'implication logique dans la définition de la stratégie (7.1). Deux cas peuvent se produire :

- Si v est une position d'Ève (○), alors $\delta(v) = \bigvee M(v)$, et l'implication (7.1) signifie $\bigwedge \sigma_{\mathrm{E}}[\pi] \Rightarrow \bigvee M(v)$, *c.-à-d.* $\sigma_{\mathrm{E}}[\pi] \cap M(v) \neq \emptyset$. Ainsi, $\sigma_{\mathrm{E}}[\pi]$ peut être comprise comme sélectionnant (au moins) un coup de $M(v)$, le coup joué par la stratégie.
- Si v est une position d'Adam (□), alors $\delta(v) = \bigwedge M(v)$, et l'implication (7.1) signifie $\bigwedge \sigma_{\mathrm{E}}[\pi] \Rightarrow \bigwedge M(v)$, *c.-à-d.* $\sigma_{\mathrm{E}}[\pi] = M(v)$. Ainsi, une stratégie pour Ève se doit d'envisager tous les coups possibles issus de v.

Par dualité, une **stratégie pour Adam** est un ensemble clos de parties σ_{A} tel que pour toute partie partielle $\pi \in \mathit{pref}(\sigma_{\mathrm{A}})$ se terminant à la position v,

$$\delta(v) \quad \Rightarrow \quad \bigvee \sigma_{\mathrm{A}}[\pi]\ .$$

Lemme 7.3. *Tout jeu a au moins une partie. Chaque joueur possède au moins une stratégie.*

Démonstration. Le seul point pouvant poser problème serait qu'une position ne soit pas l'origine d'un coup. Or, les combinaisons booléennes sont non-triviales par définition, ce qui interdit ce cas. Ainsi, il est possible de construire une partie par récurrence. La récurrence débute avec la partie partielle vide (commençant et se terminant en v_0). À chaque étape, un nouveau coup est ajouté (il existe d'après la remarque précédente). Ce processus peut se terminer par la production d'un coup final, formant ainsi une partie finie. Sinon, le processus se déroule une infinité de fois, produisant une partie infinie.

Du fait de l'existence d'une partie, l'ensemble des parties est non-vide. Il s'agit de plus d'un ensemble clos, et qui satisfait bien évidemment l'implication (7.1) (car $\delta(v)$ est différent de «vrai»). L'ensemble des parties forment donc une stratégie valide pour Ève. CQFD

Il est standard que l'affrontement de deux stratégies produise exactement une partie. Ici, notre notion de stratégie est plus faible (elle correspond aux quasi-stratégies) et donc l'affrontement de deux stratégies résulte en au moins une partie (non nécessairement unique).

Lemme 7.4. *Si σ_{E} est une stratégie pour Ève et σ_{A} est une stratégie pour Adam, alors $\sigma_{\mathrm{E}} \cap \sigma_{\mathrm{A}} \neq \emptyset$.*

Démonstration. Remarquons que pour toute partie partielles $\pi \in \mathit{pref}(\sigma_{\mathrm{E}}) \cap \mathit{pref}(\sigma_{\mathrm{A}})$ se terminant en v, nous avons

$$\bigwedge \sigma_{\mathrm{E}}[\pi] \quad \Rightarrow \quad \delta(v) \quad \Rightarrow \quad \bigvee \sigma_{\mathrm{A}}[\pi] \ .$$

Il s'ensuit que $\sigma_{\mathrm{E}}[\pi] \cap \sigma_{\mathrm{A}}[\pi] \neq \emptyset$. Soit donc m un élément de cette intersection, il satisfait $\pi m \in \mathit{pref}(\sigma_{\mathrm{E}}) \cap \mathit{pref}(\sigma_{\mathrm{A}})$.

En répétant cet argument de prolongation, soit une partie finie est produite, appartenant à la fois à σ_{E} et σ_{A}, sinon une partie infinie π est produite. Comme les stratégies sont topologiquement closes, de nouveau, π appartient à la fois à σ_{E} et σ_{A}. CQFD

Une stratégie **gagnante** (pour Ève) est une stratégie telle que toutes les parties qui la composent sont gagnantes (pour Ève). Une conséquence directe du lemme 7.4 est qu'il est impossible que Ève et Adam aient simultanément une stratégie gagnante. En effet, supposons que cela soit le cas, alors il existe une partie à la fois dans la stratégie d'Ève et dans la stratégie d'Adam. Cette partie est soit gagnante pour Ève, soit gagnante pour Adam. Supposons qu'elle le soit pour Ève, alors cela contredirait le fait que la stratégie d'Adam est gagnante. Ainsi, au plus un joueur possède une stratégie gagnante.

La question opposée est la plus intéressante : existe-t-il une stratégie gagnante pour l'un des joueurs. La réponse est bien connue : en général non. En effet il est possible de construire (en utilisant l'axiome du choix) une condition de gain W et un jeu (fini) tel qu'aucun des joueurs n'a de stratégie gagnante. En revanche, pour les jeux de durée finie (toutes les parties sont finies), l'un des joueurs possède une stratégie gagnante. Ce résultat s'étend à des cas plus compliqués. Ainsi, pour des conditions de gain «topologiquement raisonnables», l'un des joueurs a toujours une stratégie gagnante. Il s'agit du fameux théorème de détermination borélienne de Martin.

Théorème 7.5 (Martin [68]). *Dans les jeux à condition de gain borélienne, l'un des joueurs a une stratégie gagnante.*

Ainsi, pour tout jeu (de condition borélienne), exactement un des joueurs a une stratégie gagnante. Ce joueur est déclaré vainqueur du jeu. Les jeux utilisés en théorie des automates utilisent en général des conditions de gain bien particulières. Celles que nous utiliserons sont présentées dans la section suivante.

7.2 Conditions de base et leur mémoire

En théorie des automates, certaines conditions de gain bien particulières sont considérées, il s'agit des conditions de Büchi, de parité, de Rabin, de Streett ou de Muller. L'un de leurs intérêts est que, dans un jeu utilisant ces conditions, il est possible de connaître avec précision la mémoire nécessaire pour que le vainqueur du jeu puisse implémenter sa stratégie. Ce paramètre de mémoire est crucial dans l'application de ces jeux aux automates. Nous présentons ces résultats au cours de ce chapitre.

Fixons tout d'abord les conditions de gain auxquelles nous nous intéresserons, ainsi que quelques notations.

Condition d'accessibilité Une condition d'accessibilité assure que toutes les parties atteignent une position gagnante. Dans notre cas, l'alphabet ne contient qu'une lettre d'arité 1, «−» et deux lettres d'arité 0, «vrai» et «faux». Les mots gagnants sont ceux se terminant par vrai. Les autres mots, et en particuliers les mots infinis, sont perdants. L'objectif du joueur Ève est donc d'atteindre une position lui permettant de jouer le terminateur vrai. Les conditions d'accessibilité sont simplement dénotées `accessibilité`.

Condition de sûreté Une condition de sûreté est la condition duale de la condition de sûreté (en échangeant le rôle de vrai et faux). Elle caractérise les jeux où l'on cherche à éviter un événement mauvais. L'alphabet est le même, mais cette fois-ci, les mots gagnants sont ceux se terminant par vrai ainsi que les mots infinis. L'objectif d'Ève est donc d'éviter d'atteindre faux. Les conditions de sûreté sont notées `sûreté`.

Condition de Büchi La condition de Büchi requiert quant à elle qu'un bon événement apparaisse infiniment souvent. L'alphabet contient donc cette fois deux lettres d'arité 1, «1» et «2» (2 étant «bon» et 1 «mauvais»), et les terminateurs sont toujours vrai et faux. Comme toujours les mots se terminant par vrai sont gagnants, et ceux se terminant par faux sont perdant. Parmi les mots infinis, seuls ceux possédant une infinité de lettre 2 sont gagnants. Les conditions de Büchi sont notées `Büchi`.

Condition de co-Büchi La condition de co-Büchi requiert quant à elle qu'un bon événement apparaisse ultimement. L'alphabet contient donc cette fois deux lettres d'arité 1, «0» et «1» (0 étant «bon» et 1 «mauvais»), et les terminateurs sont toujours vrai et faux. Parmi les mots infinis, seuls ceux possédant un nombre fini d'occurrences de 1

sont gagnants. Comme les mots se terminant par vrai sont gagnants, et ceux se terminant par faux sont perdant. À renommage des lettres prêt, il s'agit du complément de la condition de Büchi. Les conditions de co-Büchi sont notées `co-Büchi`.

Condition de Muller La condition de Muller utilise un nombre fini de lettres d'arité 1, appelées **couleurs**, et toujours les terminateurs vrai et faux. La condition est décrite au moyen d'un ensemble d'ensembles de couleurs M. Un mot est accepté si, soit il est fini et se termine par vrai, soit il est infini, et l'ensemble des couleurs apparaissant infiniment souvent dans le mot appartient à M. Le plus souvent, la condition est exprimée au moyen d'une formule logique portant sur l'apparition des couleurs. Par exemple, sur les couleurs a, b, c, $\infty(a) \wedge \infty(b) \wedge \neg\infty(c)$ signifie qu'une partie est gagnante si une infinité de a est rencontrée, une infinité de b est rencontrée, et seulement un nombre fini de c. Les conditions de Müller sont simplement notées `Muller`.

Condition de Rabin La condition de Rabin utilise un ensemble fini de couleurs (il s'agit d'un cas particulier de Muller), et les mots gagnants sont ceux qui satisfont une disjonction de propriétés de la forme «les couleurs dans F apparaissent un nombre fini de fois et les couleurs dans I infiniment souvent» où F et I sont des ensembles de couleurs. Une telle paire (I, F) est appelée une **paire de Rabin**. Les conditions de Rabin sont simplement notées `Rabin`.

Condition de Streett La condition de Streett est le complément (à échange des terminateurs vrai/faux près) de la condition de Rabin. Il s'agit cette fois de satisfaire une conjonction de propriétés de la forme «si une couleur dans Q apparaît infiniment, alors une couleur dans R apparaît infiniment souvent». Une telle paire (Q, R) s'appelle une **paire de Streett**. Les conditions de Streett sont simplement dénotées `Streett`.

Condition de parité La condition de parité utilise un intervalle fini $[i, j]$ d'entiers (appelés **priorités**) comme alphabet d'arité 1, et les terminateurs vrai et faux. Un mot est gagnant s'il se termine par vrai ou bien s'il est infini et que la plus grande priorité apparaissant infiniment souvent est paire. Pour cette définition, la condition de Büchi correspond au cas de l'intervalle de priorités $[1, 2]$, et la condition de co-Büchi à l'intervalle de priorités $[0, 1]$. La condition d'accessibilité correspond à une unique priorité 1, et la condition de sûreté à une unique priorité 0. Les conditions de parité sont simplement notées `parité`.

Une condition de parité peut se voir comme une condition de Rabin, en effet, il s'agit de trouver un entier $n \in [i, j]$ pair tel que n apparaît infiniment et les couleurs au dessus apparaissent seulement un nombre fini de fois. Une condition de parité peut aussi être vue comme une condition de Streett. En effet, il s'agit, pour toute couleur impaire $n \in [i, j]$ apparaissant infiniment souvent, de voir une couleur supérieure paire apparaître aussi infiniment souvent. En fait, les conditions de parité peuvent être vues exactement comme les conditions qui sont à la fois de Rabin et de Streett.

Condition fermée Il s'agit d'une condition de gain W topologiquement close. Étant donné un langage de mots avec terminateurs L, sa clôture topologique est dénotée $\overrightarrow{L}$.

Ces conditions sont aussi parfois appelées «de sûreté». Nous prenons la convention dans ce travail de n'utiliser le terme de sûreté que pour la condition duale de l'accessibilité (*cf.* ci-dessus).

Les conditions mentionnées ci-dessus ont des propriétés importantes. Tout d'abord, les jeux finis sont décidables (sauf pour les conditions fermées qui peuvent être très complexes d'un point de vue calculatoire).

En théorie des automates, le point crucial dans l'étude des jeux est d'établir des résultats concernant la quantité de mémoire nécessaire au vainqueur du jeu pour atteindre son objectif. C'est également le cas pour les fonctions de coût régulières sur les arbres, et ce même dans l'études des arbres finis (en ce sens, les deux théories diffèrent grandement).

Nous avons vu la notion de stratégie. Il s'agit maintenant de définir la quantité de mémoire utilisée par une stratégie. Étant données une stratégie σ_{E} pour Ève et une partie partielle $\pi \in \mathit{pref}(\sigma_{\mathrm{E}})$, dénotons par $\pi^{-1}\sigma_{\mathrm{E}}$ l'ensemble $\{\pi' \ : \ \pi\pi' \in \sigma_{\mathrm{E}}\}$, *c.-à-d.* la stratégie décrivant comment continuer après avoir joué π. La **mémoire** utilisée par σ_{E} à la position v est alors

$$|\{\pi^{-1}\sigma_{\mathrm{E}} \ : \ \pi \in \mathit{pref}(\sigma_{\mathrm{E}}) \text{ partie partielle se terminant en } v\}| \ .$$

La **mémoire de la stratégie** est alors le suprémum sur toutes les positions v de la mémoire utilisée à la position v. Une **stratégie sans mémoire**, ou **positionnelle** est une stratégie de mémoire 1. Ceci signifie que si deux parties partielles π, π' dans σ_{E} aboutissent à la même position, alors $\pi^{-1}\sigma_{\mathrm{E}} = (\pi')^{-1}\sigma_{\mathrm{E}}$, c'est-à-dire que la stratégie continue de manière identique quelle que soit la partie partielle qui a abouti à la position v.

On dira qu'une condition de gain est **de mémoire** k si dans tous les jeux sur cet objectif tels que le joueur Ève gagne, alors Ève gagne avec une mémoire au plus k. Soulignons bien que dans tout ce document, à chaque fois que la notion de mémoire est considérée sans autre précision, il s'agit bien de la mémoire nécessaire pour le joueur Ève uniquement.

La proposition suivante énonce quelques résultats fondamentaux de la théorie classique concernant la mémoire des conditions usuelles de gain.

Proposition 7.6 ([54, 108])**.** *Les conditions de Rabin (et donc de parité) sont sans mémoire. Une condition de Muller à m couleurs est à mémoire $m!$. Les conditions ω-régulières sont à mémoire finie.*

Un résultat nous servira par la suite. Il s'agit d'un cas simple d'une collaboration en cours avec Nathanaël Fijalkow et Florian Horn dont l'objectif est de comprendre finement la mémoire nécessaire dans un jeu.

Proposition 7.7. *Les conditions régulières fermées sont à mémoire le suprémum des tailles des antichaînes de résidus pour l'inclusion, c'est à dire,*

$$\sup\{n \ : \ \textit{il existe } u_1,\ldots,u_n \textit{ tels que } u_i^{-1}W \not\subseteq u_j^{-1}W \neq \emptyset \textit{ et } u_j^{-1}W \not\subseteq u_i^{-1}W \neq \emptyset \textit{ pour tout } i \neq j\}$$

En particulier, une condition fermée est sans mémoire si et seulement si ses résidus sont totalement ordonnés pour l'inclusion.

En fait, nous utiliserons la conséquence suivante.

Fait 7.8. Soit L un langage régulier dont les résidus sont totalement ordonnés, alors sa clôture topologique $\overrightarrow{L}$ est sans mémoire.

7.3 Objectifs et jeux de coût

Nous passons maintenant aux jeux de coût qui sont spécifiques à ce travail. Le principe des jeux de coût est de remplacer la notion de condition de gain par une notion quantitative, celle d'objectif. Un jeu de coût, au lieu d'être gagné par Adam ou Ève, produit une valeur entière ou infinie, représentant le «coût» pour Ève de gagner ce jeu. Les définitions sont assez standards, si ce n'est en ce qui concerne le domaine $\mathbb{N} \cup \{\infty\}$.

Un **objectif** est un triplet $O = (\mathbb{C}, f, opt)$ dans lequel :
- $\mathbb{C}$ est un alphabet avec terminateur d'**actions**, les lettres dans $\mathbb{C}_0$ étant appelées **actions finales**,
- f est une **application** des mots sur $\mathbb{C}$ dans $\mathbb{N} \cup \{\infty\}$,
- $opt \in \{\min, \max\}$ est le paramètre d'**optimisation**.

L'intuition est que dans le contexte d'un jeu, un objectif assigné à un joueur (en pratique Ève) décrit quelle fonction le joueur a pour objectif d'optimiser («f»), sur quel domaine est définie cette fonction («$\mathbb{C}^\infty$»), et si le joueur doit essayer de minimiser cette valeur ou de la maximiser («opt»). Nous utiliserons la même notion pour les automates (en particulier, cela permet d'unifier les notions de `B` et de `S`-automates) (voir section 7.6).

Étant donné un objectif $O = (\mathbb{C}, f, opt)$ et un entier n, posons

$$O_n = \begin{cases} \{u \in \mathbb{C}^\infty \ : \ f(u) \leq n\} & \text{si } opt = \min \\ \{u \in \mathbb{C}^\infty \ : \ f(u) \geq n\} & \text{si } opt = \max\ . \end{cases}$$

Le **dual** d'un objectif O, noté $\overline{O}$ s'obtient en échangeant les paramètres d'optimisation min et max.

Un **jeu de coût** $\mathcal{G} = (V, v_0, M, \delta, O)$ est constitué :
- d'un ensemble de **positions** V, d'une **position initiale** v_0, d'un ensemble de **coups** M et d'une **application de contrôle** δ, tous définis comme dans le cas des jeux booléens (voir page 104),
- d'un **objectif** $O = (\mathbb{C}, f, opt)$.

Un jeu de coût calcule une valeur dans $\mathbb{N} \cup \{\infty\}$ au lieu de simplement déterminer une valeur booléenne (Ève gagne/Ève perd). La valeur du jeu est définie par

$$\mathit{valeur}(\mathcal{G}) = \begin{cases} \inf\{n \ : \ \text{Ève gagne } \mathcal{G}[O_n]\} & \text{si } opt = \min \\ \sup\{n \ : \ \text{Ève gagne } \mathcal{G}[O_n]\} & \text{sinon,} \end{cases}$$

où $\mathcal{G}[W]$ dénote le jeu booléen obtenu de $\mathcal{G}$ en remplaçant l'objectif O par la condition de gain W.

Le **dual** d'un jeu de coût s'obtient en dualisant l'application de contrôle ainsi que l'objectif. Cela revient à inverser le rôle des joueurs. Le dual d'un jeu de coût $\mathcal{G}$ est noté $\overline{\mathcal{G}}$. Le résultat de détermination borélienne de Martin (théorème 7.5) que nous avons vu au cours de la section précédente s'adapte comme suit au cas quantitatif.

Proposition 7.9. *Si $\mathcal{G}$ est un jeu de coût borélien (c.-à-d., si O_n est borélien pour tout n), alors*

$$\mathit{valeur}(\mathcal{G}) = \mathit{valeur}(\overline{\mathcal{G}}) \ .$$

Démonstration. Supposons, sans perdre de généralité, que l'objectif de $\mathcal{G}$ est de minimiser, sinon, il suffit de remplacer le jeu par son dual.

Remarquons tout d'abord (*monotonicité*) que si $m \leq n$, alors $O_m \subseteq O_n$, et en conséquence, si Ève gagne $\mathcal{G}[O_m]$ alors Ève gagne aussi $\mathcal{G}[O_n]$, avec la même stratégie.

Soit maintenant $m = \mathit{valeur}(\mathcal{G})$ et $n = \mathit{valeur}(\overline{\mathcal{G}})$. Supposons m, n finis. Cela signifie que m est l'entier minimal tel que Ève gagne le jeu $\mathcal{G}[O_m]$, et n et l'entier maximal tel qu'Adam gagne le jeu $\mathcal{G}[O_{n-1}]$.

Supposons $m < n$, et donc $m \geq n - 1$. Cela signifie d'une part qu'Ève gagne $\mathcal{G}[O_m]$ et donc d'après le remarque de monotonicité Ève gagne aussi $\mathcal{G}[O_{n-1}]$, et d'autre part qu'Adam gagne $\mathcal{G}[O_{n-1}]$. Contradiction.

Supposons maintenant que $n > m$. Alors, Ève ne gagne pas le jeu $\mathcal{G}[O_{m-1}]$ et donc par l'argument de monotonicité ne gagne pas le jeu $\mathcal{G}[O_n]$ non plus. Comme par ailleurs Adam ne gagne pas le jeu $\mathcal{G}[O_n]$, et que O_n est borélien, cela contredit le théorème de Martin 7.5.

Les cas correspondants à m ou n infini sont similaires. CQFD

Remarquons enfin que les objectifs et les jeux peuvent être étudiés sous l'angle des fonctions de coût, ce que nous ferons par la suite. Dans notre cas, si les objectifs sont équivalents, alors les valeurs des jeux sont elles aussi équivalentes. Notons $O \preccurlyeq_\alpha O'$ pour signifier que les paramètres d'optimisation des objets O et O' sont les mêmes, et que leurs fonctions respectives f, f' satisfont $f \preccurlyeq_\alpha$. La proposition suivante formalise ce point.

Proposition 7.10. *Soient $O \preccurlyeq_\alpha O'$ deux objectifs et soit $\mathcal{G}$ une arène alors*

$$\mathit{valeur}(\mathcal{G}[O]) \preccurlyeq_\alpha \mathit{valeur}(\mathcal{G}[O']) \ .$$

Démonstration. Considérons le cas d'objectifs à minimiser. Remarquons que $O \preccurlyeq_\alpha O'$ signifie alors que $O'_n \subseteq O_{\alpha(n)}$ pour tout n. Supposons maintenant que $\mathit{valeur}(\mathcal{G}[O']) = n$, cela signifie qu'Ève gagne le jeu $\mathcal{G}[O'_n]$. Comme $O'_n \subseteq O_{\alpha(n)}$, Ève gagne $\mathcal{G}[O_{\alpha(n)}]$ en utilisant la même stratégie. Ainsi, $\mathit{valeur}(\mathcal{G}[O]) \leq \alpha(n)$. CQFD

7.4 Les objectifs de base

Dans la suite, un certain nombre d'objectifs de base seront utilisés. En particulier, nous retrouverons les notions de B-automates et de S-automates ainsi que les conditions infinitaires de parité, de Rabin, de Büchi, de Streett et de Muller. Il s'agit au cours de cette section de mettre en place une terminologie permettant de composer les objectifs et de faire référence avec précision à chacun de ces cas.

Objectifs infinitaires Nous avons vu dans le cadre des jeux booléens les condition ω-régulières, de sûreté, d'accessibilité, de Muller, de Rabin, de Streett, ou de parité. Ces conditions peuvent être vues comme des objectifs utilisés dans des jeux de coût.

Ainsi, soit W une condition de gain et $\mathcal{G}$ le jeu de coût d'objectif W. Alors il est aisé de vérifier que $\mathit{valeur}(\mathcal{G}) = 0$ si et seulement si Ève gagne le jeu $\mathcal{G}[W]$, et les stratégies gagnantes sont les mêmes. Ainsi, les résultats de mémoire dans les jeux booléens se transfèrent directement à ce cas. Nous donnons ainsi un sens aux objectifs `accessibilité`, `sûreté`, `Büchi`, `co-Büchi`, `Muller`, `Rabin`, `Streett` et `parité`.

L'objectif `B` (sur k compteurs) est un objectif à minimiser. L'ensemble des lettres d'arité 1 (sur les compteurs Γ) est $\{\epsilon, \mathtt{ic}, \mathtt{r}\}^\Gamma$ (nous utiliserons les mêmes notations que pour les automates). L'ensemble des lettres d'arité 0 est $\{0, \infty\}$. La fonction à optimiser $\mathit{coût}_{\mathtt{B}}$ est telle que $\mathit{coût}_{\mathtt{B}}(u) \leq n$ si u ne se termine pas par ∞ et les compteurs ne dépassent jamais la valeur n, quand les lettres $\{\epsilon, \mathtt{ic}, \mathtt{r}\}^\Gamma$ sont évaluées comme pour les automates.

En reprenant les notations des automates,

$$\mathit{coût}_{\mathtt{B}}(u) = \begin{cases} \infty & \text{si } u \text{ se termine par } \infty, \\ \mathit{coût}_{\mathtt{B}}(v) & \text{si } u = v0, \\ \sup_{v \sqsubset u} \mathit{coût}_{\mathtt{B}}(v) & \text{si } u \text{ est infini.} \end{cases}$$

Ainsi, l'utilisation du terminateur ∞ permet d'annuler le calcul et d'obtenir une valeur infinie quel que soit ce qui précède. Une autre façon de décrire cette condition est au moyen d'une clôture topologique :

$$(\mathit{coût}_{\mathtt{B}})_n = \overrightarrow{\{u \ : \ \mathit{coût}_{\mathtt{B}}(u) \leq n\}0} \ .$$

Dans le cas où les compteurs sont hiérarchisés, nous parlerons de l'objectif `hB`. Dans le cas d'un unique compteur, sans remise à 0, nous parlerons de l'objectif `distance`.

L'objectif $\neg$`B` (sur k compteurs) est un objectif à maximiser. *Il ne s'agit pas du dual de l'objectif* `B`, qui lui est noté $\overline{\mathtt{B}}$). L'ensemble des lettres d'arité 1 (sur k compteurs) est $\{\epsilon, \mathtt{ic}, \mathtt{r}\}^\Gamma$ (on utilisera la encore les mêmes notations que pour les automates). L'ensemble des lettres terminateurs est $\{0, \infty\}$. La fonction de coût à optimiser est

décrite (par réduction au cas des automates) comme :

$$co\hat{u}t_{\neg \mathtt{B}}(u) = \begin{cases} \infty & \text{si } u \text{ se termine par } \infty \text{ ou } u \text{ est infini} \\ co\hat{u}t_B(v) & \text{si } u = v0 \end{cases}$$

Ainsi, l'objectif d'Ève est d'éviter que la partie atteigne un terminateur 0 alors qu'aucun des compteurs n'a dépassé la valeur n. Dans l'autre sens, cela signifie que l'objectif d'Ève est, soit d'atteindre le terminateur ∞, soit que l'un des compteurs dépasse la valeur n, soit que la partie soit infinie. Là encore, une autre façon de décrire cette condition est au moyen d'une clôture topologique :

$$(co\hat{u}t_{\neg \mathtt{B}})_n = \overrightarrow{\{u \ : \ co\hat{u}t_{\mathtt{B}}(u) \geq n\}0 \cup (\{\epsilon, \mathtt{ic}, \mathtt{r}\}^\Gamma)^*0}$$

Soulignons qu'il ne s'agit pas de la même fonction à optimiser que l'objectif `B`*, et par consquent, il ne s'agit pas du dual de l'objectif* `B`*. Les objectifs* $\neg$`B` *et* $\overline{\mathtt{B}}$ *se comportent de manière identique sur les mots finis. Leur différence n'apparaît que sur les mots finis.* Plus précisément, la condition $\overline{\mathtt{B}}$ peut être comprise comme la condition $\neg\mathtt{B} \wedge$ `accessibilité`, et la condition $\overline{\neg\mathtt{B}}$ peut être vue comme la condition $\mathtt{B} \wedge$ `accessibilité`, par simple renommage des alphabets.

Dans le cas ou les compteurs sont hiérarchisés, nous parlerons de l'objectif $\neg$`hB`. Dans le cas d'un unique compteur, sans remise à 0, nous parlerons de l'objectif $\neg$`distance`.

L'objectif `S` Il s'agit d'un objectif à maximiser. L'ensemble des lettres d'arité 1 (sur k compteurs) est $\{\varepsilon, \mathtt{i}, \mathtt{cr}, \mathtt{r}\}^\Gamma$ (on utilisera la encore les mêmes notations que pour les automates). L'ensemble des terminateurs est $\{0, \infty\}$. La fonction de coût à optimiser est décrite (par réduction au cas des automates) par :

$$co\hat{u}t_{\mathtt{S}}(u) = \begin{cases} co\hat{u}t_S(u) & \text{si } u \text{ est infini} \\ co\hat{u}t_S(v) & \text{si } u = v\infty \\ 0 & \text{si } u = v0 \end{cases}$$

Ainsi, l'objectif, pour que la valeur d'une séquence soit au moins n est d'éviter de terminer par 0, et de garantir que lorsque testés, tous les compteurs aient une valeur au moins égale à n.

Là encore, il s'agit d'un objectif topologiquement clos car :

$$(co\hat{u}t_{\mathtt{S}})_n = \overrightarrow{\{u\infty \ : \ co\hat{u}t_{\mathtt{S}}(u) \geq n\}} \ .$$

Objectif dual Si un objectif O peut être utilisé, alors son dual $\overline{O}$ peut l'être aussi.

La *disjonction* de deux objectifs de même paramètre d'optimisation. Soient $O = (\mathbb{A}, f, opt)$ et $O' = (\mathbb{B}, g, opt)$. Alors, si $opt = \min$, $O \vee O'$ dénote l'objectif $(\mathbb{A}\times\mathbb{B}, \min(f, g), \min)$, et si $opt = \max$ alors $O \vee O'$ dénote l'objectif $(\mathbb{A} \times \mathbb{B}, \max(f, g), \max)$.

Cette dissymétrie provient du fait que nous considérons le gain du point de vue du joueur Ève. Ainsi, dans le cas des objectifs booléens, l'union des objectifs correspond bien au minimum.

Remarquons que dans le produit d'alphabet, il n'est pas autorisé de former la paire d'un terminateur avec un non-terminateur. Ainsi, $\mathbb{A} \times \mathbb{B} = (\mathbb{A}_1 \times \mathbb{B}_1, \mathbb{A}_0 \times \mathbb{B}_0)$.

La *conjonction* de deux objectifs de même paramètre d'optimisation. Soient $O = (\mathbb{A}, f, opt)$ et $O' = (\mathbb{B}, g, opt)$. Alors, si $opt = \min$, $O \wedge O'$ dénote l'objectif $(\mathbb{A}\times\mathbb{B}, \max(f, g), \min)$, et si $opt = \max$ alors $O \wedge O'$ dénote l'objectif $(\mathbb{A} \times \mathbb{B}, \min(f, g), \max)$.

En utilisant cette terminologie, nous verrons comment résoudre les jeux `distance`$\wedge$`Rabin` ou les jeux `hB`$\wedge$`Büchi`, etc... Soulignons qu'il convient d'être vigilant concernant les notations. Tout n'est pas connu sur ces jeux, et en particulier échanger $\neg$`B` par $\overline{\texttt{B}}$ dans un énoncé peut l'invalider.

Remarque 7.11. L'usage des terminateurs est très souvent pénible, et nous omettons souvent de les traiter dans le détail. Nous justifions la possibilité de faire ce type de simplifications dans cette remarque. Pour éliminer un terminateur, il suffit de remarquer qu'il peut être remplacé par une suite infinie de lettres non-terminateurs. Considérons par exemple la condition `B`, alors pour tout mot u, $co\hat{u}t_{\texttt{B}}(u0) = co\hat{u}t_{\texttt{B}}(u(\varepsilon^\Gamma)^\omega)$. Ainsi, remplacer chaque occurrence du terminateur 0 par une boucle étiquetée par ε^Γ ne change pas la valeur du jeu. Cette technique d'élimination s'étend à la plupart des cas :

- pour la condition d'accessibilité, faux équivaut à $-^\omega$, et il ne reste donc que le terminateur vrai,
- pour la condition de sûreté, vrai équivaut à $-^\omega$, et il ne reste donc que le terminateur faux,
- dans toutes les autres conditions de Müller (autres que d'accessibilité ou de sûreté), tous les terminateurs peuvent être éliminés, et c'est en particulier le cas pour les conditions `Rabin`, `Streett`, `Büchi` et `co-Büchi`,
- pour la condition `B`, le terminateur 0 peut être remplacé par $(\varepsilon^\Gamma)^\omega$, et le terminateur ∞ peut être remplacé par $(\texttt{ic}^\Gamma)^\omega$, il ne reste donc aucun terminateur,
- pour la condition `S`, le terminateur 0 peut être remplacé par $(\texttt{rc})^\omega$, et le terminateur ∞ par $(\varepsilon^\Gamma)^\omega$,
- pour la condition $\neg$`B`, le terminateur ∞ peut être remplacé par $(\varepsilon^\Gamma)^\omega$, mais, en revanche, *rien ne permet de se débarrasser du terminateur* 0. Pour cette raison, le terminateur 0 de la condition $\neg$`B` est le seul auquel nous prêtons réellement attention dans la suite de ce travail. Nous le noterons parfois $0_{\neg\texttt{B}}$ afin de l'identifier clairement.

Enfin, il arrive que nous ayons à allonger ou raccourcir la longueur d'exécution d'automates. Il s'agit de la motivation de la notion d'élasticité. Nous dirons qu'un objectif $O = (\mathbb{C}, f, opt)$ a la propriété d'**élasticité** s'il existe un non-terminateur $\top_1$ et un terminateur $\top_0$ tels que pour tout mot partiel u et tout mot v,

$$f(uv) \leq f(u\top_0) = f(u\top_1\top_0) = f(u\top_1^\omega) \ .$$

Ainsi, la valeur de tout mot v est majorée par $\top_0, \top_1\top_0, \top_1\top_1\top_0, \dots \top_1^\omega$, et ce dans tout contexte u. Tous les objectifs ci-dessus, sauf `accessibilité` ont la propriété d'élasticité. La propriété d'élasticité est aussi préservée par $\wedge$ et $\vee$.

7.5 Étude de la structure des stratégies gagnantes

Tout comme en théorie des automates classique, la structure des stratégies, et en particulier la quantité de mémoire qu'elles nécessitent, est un point crucial dans le développement des fonctions de coût. Dans cette section, nous rassemblons tous les résultats connus concernant les conditions de coût à mémoire finie. Cette section débute par l'introduction de la notion d'$\approx$-mémoire, qui est la notion de mémoire qui nous intéressera par la suite. Les bornes supérieures connues sur cette quantité sont présentées ensuite.

Un objectif O est **à mémoire** k si O_n est à mémoire k pour tout n. Cette notion n'est pourtant pas celle qui nous intéresse. En effet, les fonctions de coût sont considérées modulo la relation $\approx$. La bonne notion de mémoire, moins restrictive, que nous introduisons maintenant est celle d'$\approx$-mémoire. Un objectif O est **à $\approx_\alpha$-mémoire** k, où α est une fonction de correction, si

- si l'objectif est à minimiser, alors pour toute arène $\mathcal{G}$ et tout entier n, si Ève gagne $\mathcal{G}[O_n]$, alors Ève gagne $\mathcal{G}[O_{\alpha(n)}]$ au moyen d'une stratégie à mémoire k,
- si l'objectif est à maximiser, alors pour toute arène $\mathcal{G}$ et tout entier n, si Ève gagne $\mathcal{G}[O_{\alpha(n)}]$, alors Ève gagne $\mathcal{G}[O_n]$ au moyen d'une stratégie à mémoire k.

Ainsi, la stratégie utilisant peu de mémoire peut gagner, mais dans une situation «plus facile» (contrôlée par α) en utilisant une mémoire au plus k. On dira parfois simplement **$\approx$-mémoire** sans rendre explicite la fonction de correction α. Bien entendu, si un objectif est à mémoire k, alors il est à $\approx$-mémoire k. De plus, si un objectif est à $\approx$-mémoire k, alors tout objectif $\approx$-équivalent est aussi à $\approx$-mémoire k (il s'agit d'une variante de la proposition 7.10).

Remarquons que, pour les objectifs quantitatifs que nous considérons, il n'est pas vrai que le vainqueur puisse toujours gagner au moyen d'une stratégie à mémoire bornée (par exemple une condition `S` nécessite une quantité infinie/non-bornée de mémoire). Cette situation diffère grandement des conditions ω-régulières. En effet toute condition ω-régulière est à mémoire finie (voir le théorème 7.6). La technique des automates déterministe en histoire développée dans la suite de ce chapitre permet de contourner cette difficulté, et de néanmoins être capable de comprendre la structure des stratégies de certains objectifs comme `S`.

L'objectif de cette section est différent. Il s'agit d'énoncer les résultats connus sur la $\approx$-mémoire. Certains résultats sont tout simplement hérités de la théorie classique. Ainsi, une condition de parité vue comme un objectif, est sans mémoire (et par conséquent $\approx$-sans mémoire). Les résultats qui suivent sont plus spécifiques aux fonctions de coût.

Théorème 7.12 ([31]). *Les objectifs* $\mathtt{hB}$ *et* $\neg\mathtt{B}$ *sont* $\approx$*-sans mémoire*[1].

Démonstration. La démonstration consiste à remplacer les objectifs $\mathtt{hB}$ et $\neg\mathtt{B}$ par des objectifs $\approx$-équivalents qui, eux, sont sans mémoire. Il s'ensuit alors par la proposition 7.10 que la condition originale est $\approx$-sans-mémoire.

Nous présentons la preuve dans le cas $\mathtt{hB}$. Soit k le nombre de compteurs impliqués dans la condition, et soit n un entier positif.

Notre point de départ est l'automate déterministe et complet naturel $\mathcal{A}_{k,n}$ qui accepte le langage $A_{k,n} = \{u \in \mathtt{Act}_{\mathtt{hB}}^* \ : \ \mathit{coût}_{\mathtt{hB}}(u) \leq n\}$. Son ensemble d'état est

$$Q = \{0, 1, \ldots, n\}^k \cup \{\infty\}$$

(nous nous permettrons d'appeler les différentes composantes d'un tel état des compteurs). L'état initial est $(0, \ldots, 0)$. Lorsqu'une action $a \in \mathtt{Act}_{\mathtt{hB}}$ est lue depuis l'état $p \in Q$, l'automate se rend dans l'état $\delta_{\mathcal{A}_{k,n}}(p, a) = q$ défini par :

1. si $p = \infty$, alors $q = \infty$,
2. sinon, si $a = \mathtt{R}_\ell$, $q = (\overbrace{0, \ldots, 0}^{\ell}, p_{\ell+1}, \ldots, p_k)$,
3. enfin, dans le cas $p \neq \infty$ et $a = \mathtt{IC}_\ell$,
   - si $p_\ell = n$, alors $q = \infty$,
   - sinon $q = (\overbrace{0, \ldots, 0}^{\ell-1}, p_\ell + 1, p_{\ell+1}, \ldots, p_k)$.

Enfin, tous les états sont acceptants sauf ∞. Il est clair que cet automate reconnait le langage $A_{k,n}$ à k compteurs.

La condition de gain $A_{k,n}$ n'est malheureusement pas positionnelle. Nous introduisons pour cette raison un langage différent, reconnu par le nouvel automate $\mathcal{B}_{k,n}$. Cet automate est en tous points identique à $\mathcal{A}_{k,n}$, si ce n'est que la nouvelle fonction de transition est définie comme suit. En partant d'un état $p \in Q$, et en lisant une lettre $a \in \mathtt{Act}_{\mathtt{hB}}$ l'automate se rend dans l'état $\delta_{\mathcal{B}_{k,n}}(p, a) = q$ défini comme $\delta_{\mathcal{B}_{k,n}}(p, a) = q$ si ce n'est dans le troisième cas :

3. enfin, dans le cas $p \neq \infty$ et $a = \mathtt{IC}_\ell$,
   - si $p_\ell = p_{\ell+1} = \cdots = p_k = n$, alors $q = \infty$,
   - sinon, soit $i \geq \ell$ l'indice minimal tel que $p_i < n$,

$$q = (\overbrace{0, \ldots, 0}^{i-1}, p_i + 1, p_{i+1}, \ldots, p_k) \ .$$

1. Remarquons que les objectifs duaux $\overline{\mathtt{hB}}$ et $\overline{\neg\mathtt{hB}}$ ne sont pas $\approx$-sans mémoire, même dans le cas à un compteur. Ils le sont si les jeux sont de durée finie (puisque $\overline{\mathtt{hB}}$ et $\neg\mathtt{B}$ coïncident sur les mots finis). Nous verrons qu'ils sont positionnels dans le cas d'un compteur, en l'absence de remise à 0.

L'effet de cette modification est que lorsqu'incrémenté, le compteur ℓ atteint la valeur $n+1$, au lieu d'entraîgner immédiatement l'automate dans l'état ∞, le compteur est remis à 0, et génère une retenue qui est propagée au compteur suivant, et ainsi de suite. Ce n'est que quand le dernier compteur dépasse n que l'automate atteint l'état ∞. Cela revient à voir l'ensemble de l'état comme un unique compteur représenté en base $n+1$ représentant un entier entre 0 et $(n+1)^k - 1$.

Soit $B_{k,n}$ le langage de mots finis accepté par $\mathcal{B}_{k,n}$, et $B_{k,n,q}$ le langage reconnu par cet automate quand l'état initial est fixé à $q \in Q$. Comme toujours pour un automate déterministe et complet, les résiduels de $B_{k,n}$ sont les $B_{k,n,q}$. Montrons qu'ils sont totalement ordonnés. Pour cela, ordonnons les états lexicographiquement, ∞ étant maximal :

$$(0,0,\dots) < (1,0,\dots) < \cdots < (N-1,0,\dots) < (0,1,0,\dots) < \dots \\ \cdots < (N-1,\dots,N-1) < \infty \ .$$

Il est aisé de vérifier que pour cet ordre, toutes les transitions de l'automate $\mathcal{B}_{k,n}$ sont monotones. Ainsi, pour $a \in \mathtt{Act}_{\mathtt{hB}_k}$ et p, q deux états,

$$p \leq a \quad \text{implique} \quad \delta(p,a) \leq \delta(q,a) \ .$$

Comme de plus, les états acceptants de $\mathcal{B}_{k,n}$ sont clos vers le bas, $p \leq q$ implique $B_{k,n,p} \supseteq B_{k,n,q}$. Ainsi, les résiduels de $L_{k,n}$ sont totalement ordonnés. Il en va bien entendu de même pour le langage avec terminateur $B_{k,n}0$. Il s'ensuit, grâce au fait 7.8, que la condition

$$\overrightarrow{B_{k,n}0} \subseteq (\mathtt{Act}_{\mathtt{hB}}, \{0,\infty\})^\infty$$

est positionnelle.

Il ne nous reste plus qu'à relier la condition $\overrightarrow{B_{k,n}0}$ à l'objectif $\mathtt{hB}$ (à k compteurs). Soit $u \in \mathtt{Act}^*_{\mathtt{hB}}$. Nous établissons tout d'abord deux faits.

1. $co\hat{u}t_{\mathtt{hB}}(u) \leq n$ implique $u \in B_{n,k}$. Supposons donc $co\hat{u}t_{\mathtt{hB}}(u) \leq n$. Cela signifie que lorsque l'automate $\mathcal{A}_{k,n}$ est exécuté, aucun compteur n'est jamais incrémenté alors qu'il possède la valeur n (sinon, l'exécution atteindrait l'état puit non-acceptant ∞). Or dans ce cas, les transitions de l'automate $\mathcal{B}_{k,n}$ sont identiques à celles de l'automate $\mathcal{A}_{k,n}$. Il s'ensuit que l'automate $\mathcal{B}_{k,n}$ atteint le même état que l'automate $\mathcal{A}_{k,n}$ après avoir lu le mot u. Cet état étant différent de ∞, nous obtenons $u \in B_{n,k}$.

2. $u \in B_{k,n}$ implique $co\hat{u}t_{\mathtt{hB}}(u) < n^k$. Supposons par la contraposée que $co\hat{u}t_{\mathtt{hB}}(u) \geq n^k$. Cela signifie qu'il existe un compteur, ℓ, et un facteur v de u tel que l'action $\mathtt{IC}_\ell$ est rencontrée n^k fois dans v, et toutes les autres actions recentrée dans v ne concernent que les compteurs $1,\dots,\ell-1$. Il est aisé de vérifier que pour la fonction de transition de $\mathcal{B}_{k,n}$, et quel que soit l'état de départ, la lecture de v entraîne l'automate dans l'état ∞. En effet, si $\pi(p)$ représente l'entier écrit en base $n+1$ sur les indices $\ell,\dots,k$ de p, alors si a est une action sur les compteurs $1,\dots,\ell-1$, $\pi(\delta_{\mathcal{B}_{k,n}}(p,a)) \geq \pi(p)$, et si $a = \mathtt{IC}_\ell$, alors $\pi(\delta_{\mathcal{B}_{k,n}}(p,a)) = \pi(p)+1$. Le résultat de l'application du facteur v à partir de tout état de l'automate est donc inéluctablement d'atteindre l'état ∞. Ainsi, $u \notin B_{k,n}$.

Il s'ensuit que pour tout n,

$$\mathtt{hB}_n = \overrightarrow{A_{k,n}0} \subseteq \overrightarrow{B_{k,n}0} \subseteq \overrightarrow{A_{k,n^k}0} = \mathtt{hB}_{n^k} \ .$$

Ainsi, supposons que $\mathit{valeur}(\mathcal{G}[\mathtt{hB}]) \leq n$, cela signifie qu'Ève gagne le jeu $\mathcal{G}[\mathtt{hB}_n]$. Par conséquent Ève gagne le jeu $\mathcal{G}[\overrightarrow{B_{k,n}0}]$. Or, nous avons vu que la condition $\overrightarrow{B_{k,n}0}$ est positionnelle, donc Ève gagne le jeu $\mathcal{G}[\overrightarrow{B_{k,n}0}]$ au moyen d'une stratégie positionnelle. Cette même stratégie permet à Ève de gagner le jeu $\mathcal{G}[\mathtt{hB}_{n^k}]$. Ainsi, l'objectif $\mathtt{hB}$ est bien $\approx$-positionnel. CQFD

Ce résultat permet de déduire un résultat plus faible pour ces mêmes jeux augmentés d'une condition de Büchi.

Théorème 7.13 (Vanden Boom [12])**.** *Les objectifs* $\mathtt{hB} \wedge \mathtt{Büchi}$ *et* $\neg\mathtt{hB} \wedge \mathtt{Büchi}$ *sont à* $\approx$*-mémoire* 2 *dans les jeux de degré fini chronologiques.*

Démonstration. Donnons une intuition de la technique de preuve. Soit donc $\mathcal{G}$ un jeu d'objectif $\mathtt{hB} \wedge \mathtt{Büchi}$ (le cas $\neg\mathtt{hB} \wedge \mathtt{Büchi}$ est identique) à la fois de degré fini et chronologique.

Le principe est de partir d'une stratégie gagnante pour Ève σ_E dans $\mathcal{G}[(\mathtt{hB} \wedge \mathtt{Büchi})_n]$ (utilisant possiblement une grande mémoire), et de la transformer en une condition de mémoire 2 (pour la condition $[(\mathtt{hB} \wedge \mathtt{Büchi})_{\alpha(n)}]$ pour un α bien choisi). Supposons donc qu'Ève gagne $\mathcal{G}[(\mathtt{hB} \wedge \mathtt{Büchi})_n]$. En utilisant le lemme de König (ce qui nécessite l'hypothèse de structure chronologique et de degré fini), il est possible de découper le jeu en «tranches» telles que dans toute partie compatible avec σ_E et dans chaque tranche, au moins une transition de Büchi est rencontrée.

En utilisant ces tranches, il s'agit alors de construire un nouveau jeu $\mathcal{G}'$, cette fois d'objectif $\mathtt{hB}$, toujours gagné par Ève. Les caractéristiques du jeu $\mathcal{G}'$ sont les suivantes. Toutes les positions de $\mathcal{G}$ sont dupliquées, et existent en version verte et en version rouge. La signification de cette couleur est qu'un état est vert si une transition de Büchi a été rencontrée dans la tranche courante. Ainsi, le jeu démarre dans la version rouge de la position initiale du jeu original. Les coups du jeu sont les mêmes que dans le jeu original, à la seule différence qu'un état garde sa couleur sauf si (a) une transition de Büchi est rencontrée, auquel cas la position devient automatiquement verte, où (b) il y a un changement de tranche, auquel cas, si la position est rouge, la partie s'arrête et Ève perd, et si la position est verte, le jeu continue dans la même position, la position passant à l'état rouge. Il est aisé de vérifier que la stratégie gagnante dans le jeu original reste gagnante dans le nouveau jeu. Par contre, ce nouveau jeu a maintenant $\mathtt{hB}$ comme objectif. Il est donc possible de lui appliquer le théorème 7.12. Ève possède donc une stratégie gagnante positionnelle dans le jeu $\mathcal{G}'[\mathtt{hB}_{\alpha(n)}]$ pour un α choisi indépendamment de $\mathcal{G}$. Il est alors aisé de transformer cette stratégie positionnelle sur $\mathcal{G}'$ en une stratégie gagnante à mémoire 2 dans le jeu $\mathcal{G}[(\mathtt{hB} \wedge \mathtt{Büchi})_{\alpha(n)}]$. Le bit de mémoire de la stratégie sert à enregistrer la couleur de la position courante dans $\mathcal{G}'$. CQFD

Proposition 7.14 (non publié, avec C. Löding)**.** *Les objectifs* $\texttt{distance}\wedge\texttt{Rabin}$ *et* $\neg\texttt{distance}\wedge\texttt{Rabin}$ *sont sans mémoire.*

Démonstration. Considérons en premier le cas de l'objectif $O = \texttt{distance} \wedge \texttt{Rabin}$. Le but de Ève dans la condition $(\texttt{distance}\wedge\texttt{Rabin})_n$ est de garantir une condition de Rabin, tout en voyant au plus n-occurrences d'incréments de l'unique compteur. L'idée de la construction est très simple, il s'agit de voir la condition $(\texttt{distance} \wedge \texttt{Rabin})_n$ comme une «succession» d'au plus n conditions de parité, le changement de conditions correspondant à l'apparition d'un incrément du compteur. Entre ces événements que sont les incréments, le résultat usuel de positionnalité pour les conditions de Rabin s'applique.

A chaque position v du jeu, il est possible d'associer un entier n qui est le plus petit entier tel qu'Ève gagne le jeu en partant de la position v dans un jeu booléen de condition $(\texttt{distance} \wedge \texttt{Rabin})_n$. Soit W_n l'ensemble des positions de valeur n. La stratégie gagnante s'obtient alors par récurrence sur n, l'hypothèse de récurrence étant :

> Il existe une stratégie d'Ève sans mémoire σ_n sur $W_0 \cup W_1 \cup \ldots W_n$ qui permet de gagner pour la condition de gain $(\texttt{distance} \wedge \texttt{Rabin})_n$.

La propriété est vraie pour $n = -1$ (de manière pathologique, la condition $(\texttt{distance} \wedge \texttt{Rabin})_{-1}$ n'étant jamais vérifiée).

Soit maintenant $n \geq 0$. Gagner la condition $(\texttt{distance} \wedge \texttt{Rabin})_n$ revient à gagner la condition de Rabin sans voir jamais d'incrément du compteur, ou à atteindre un premier incrément, à partir duquel il est possible de gagner pour la condition $(\texttt{distance}\wedge\texttt{Rabin})_{n-1}$. Considérons donc le jeu (booléen) $\mathcal{G}_n$ obtenu du jeu original en :

1. se restreignant à la zone W_n,
2. rendant «gagnantes» les transitions partant de $p \in W_n$ pour aboutir en W_m pour $m < n$ (ceci s'obtient en remplaçant partout dans l'application de contrôle les atomes de la forme (p, c, q) pour $q \in W_m$ avec $m < n$ par le terminateur $(p, \text{vrai}, \bot)$),
3. rendant «perdantes» les transitions effectuant un incrément et restant dans W_n (même construction avec le terminateur faux),
4. rendant «perdantes» les transitions aboutissant dans W_m pour $m > n$ (de nouveau la même construction).

Il est clair qu'une stratégie dans le jeu original au départ d'une position $v \in W_n$ gagnante pour l'objectif $(\texttt{distance} \wedge \texttt{Rabin})_n$, est une stratégie gagnante dans $\mathcal{G}_n$ (quand elle est restreinte à W_n). Ainsi, Ève gagne depuis toutes les positions de $\mathcal{G}_n$. Comme $\mathcal{G}_n$ est un jeu de Rabin (tous les incréments ont été retirés dans la construction), il existe une stratégie sans mémoire τ_n gagnante depuis toutes les positions de W_n. Considérons maintenant la stratégie sans mémoire σ_{n-1} obtenue par hypothèse d'induction. Il suffit de construire la nouvelle stratégie σ_{n-1} consistant à commencer à jouer comme τ_n jusqu'à ce que l'on sorte de W_n (ce qui peut durer indéfiniment), ou sinon aboutit dans W_m pour $m < n$, et dans ce dernier cas continue comme σ_{n-1}. Il est clair que cette stratégie permet de gagner le jeu $\mathcal{G}[(\texttt{distance} \wedge \texttt{Rabin})_n]$ depuis toutes les positions $W_0 \cup \cdots \cup W_n$.

Le cas de l'objectif $\neg$`distance` $\wedge$ `Rabin` est similaire. CQFD

Nous avons présenté ci-dessus tous les cas connus de mémoire finie dans les jeux de coût (d'autres résultats sont en fait connus, mais ils se déduisent de ceux ci-dessus). Il reste des questions ouvertes importantes. C'est en particulier le cas de la suivante pour laquelle les techniques développées ci-dessus s'avèrent insuffisantes.

Question ouverte 7.15. *Est-ce que l'objectif* `Rabin` $\wedge$ `hB` *est* $\approx$*-sans mémoire, ou à* $\approx$*-mémoire finie ? pour des arènes d'un certain type ? pour une condition équivalente à réduction déterministe en histoire près ?*

Certaines variantes sont envisagées dans cette question ouverte. Ces versions restreintes correspondent à la possibilité d'appliquer ce résultat pour montrer la décidabilité de la logique monadique de coût sur les arbres infinis. Dans cette application, seules des arènes obtenues en exécutant un automate (non-déterministe ou alternant) sont considérées. Ces arènes sont en particulier sans cycles et de largeur arborescente bornée. Ainsi, il n'est pas nécessaire de répondre positivement à cette question ouverte dans toute sa généralité pour obtenir la décidabilité de la logique monadique de coût (certainement l'application la plus importante), mais il suffit de le faire sur des arènes d'un type particulier.

Nous verrons également la notion d'automates déterministes en histoire. Cet outil permet de traduire un objectif en un autre, de manière suffisamment profonde pour parfois changer les quantités de mémoire nécessaire (passant de «infini» à «fini» en particulier). Il convient donc d'envisager l'hypothétique existence d'un objectif qui soit meilleur que `Rabin` $\wedge$ `hB` en terme de mémoire, et qui lui soit équivalent à réduction déterministe en histoire près. Cette technique est développée dans les sections qui suivent. Il est donc possible que la condition `Rabin` $\wedge$ `hB` n'admette pas de stratégie à mémoire finie sur les arènes les plus favorable, mais qu'il soit néanmoins possible de transformer cet objectif en un autre objectif, qui, lui, admette des stratégies sans mémoire.

7.6 Automates alternants

Nous avons vu comment calculer des valeurs dans $\mathbb{N} \cup \{\infty\}$ au moyen de jeux. Les automates à coût ont également pour but de calculer ce type de valeur. Dans cette section, nous introduisons la notion d'automates alternants, dans le cas booléen, puis à coût. Les automates ainsi définis correspondent précisément, dans les cas `B` et `S`-non-déterministe, aux automates des chapitres précédents.

La nouvelle définition que nous donnons ici utilise une syntaxe très proche de celle des jeux. Cette analogie sera prolongée dans la suite de ce chapitre. Comme dans les sections précédentes nous commençons par présenter le cas booléen (section 7.6.1), pour aborder ensuite le cas des automates de coût (section 7.6.2).

7.6.1 Automates alternants booléens

La définition suit fidèlement celles des jeux (voir section 7.1, page 104). Un **automate alternant** $\mathcal{A} = (Q, \mathbb{A}, q_0, W, \delta)$ est constitué :

- d'un ensemble d'**états** Q (non nécessairement fini, pour garder l'analogie avec les jeux),
- d'un **alphabet d'entrée** avec terminateur, $\mathbb{A}$,
- d'un **état initial** q_0,
- d'une **condition de gain** $W \subseteq \mathbb{C}^\infty$ (pour un certain alphabet avec terminateur $\mathbb{C}$),
- d'une relation de transition $\Delta = \Delta_0 \cup \Delta_1$ où

$$\begin{aligned} &\Delta_0 \subseteq Q \times \mathbb{A} \times \mathbb{A}_0 \times \{\bot\}\,, \\ \text{et} \quad &\Delta_1 \subseteq Q \times \mathbb{A}_1 \times \mathbb{C}_1 \times Q\ . \end{aligned}$$

  On note $\Delta(q)$ l'ensemble des transitions issues de $q \in Q$, c'est à dire $\{(q,a,c,r) : (q,a,c,r) \in \Delta\}$, et $\Delta(q,a)$ l'ensemble des transitions de la forme (q,a,c,r).
- d'une **application de contrôle** δ qui à $q \in Q$ et $a \in \mathbb{A}$ associe $\delta(q,a) \in \mathcal{B}^+(\Delta(q,a))$.

Remarque 7.16 (cas des jeux). Remarquons que cette définition coïncide avec celles des jeux, dans le cas ou l'alphabet en entrée de l'automate ne contient qu'une lettre d'arité 1, et aucun terminateur, *c.-à-d.* que l'alphabet en entrée est $\mathbb{I}$. Il n'existe qu'un unique mot qui peut être formé sur cet alphabet. Dans cette analogie, l'automate accepte cet unique mot si et seulement si le jeu est gagné par Ève.

Commençons par montrer comment un tel automate peut être utilisé pour accepter ou rejeter un mot en entrée. Cela s'obtient très facilement : en effectuant le produit du mot avec l'automate, nous obtenons un jeu. Le mot est accepté par l'automate si ce jeu est gagné par Ève. Ainsi, considérons un mot $u = a_0 a_1 \dots$ (possiblement infini, ou se terminant par un terminateur) sur l'alphabet $\mathbb{A}$. Nous construisons alors le jeu $\mathcal{G}(\mathcal{A}, u)$ de positions $\{0, 1, \dots\} \times Q$ (la première composante correspondant aux indices des lettres dans le mot u). La position initiale de $\mathcal{G}(\mathcal{A}, u)$ est $(0, q_0)$, l'objectif est celui de l'automate, et l'application de contrôle $\delta_{\mathcal{G}(\mathcal{A},u)}$ est définie pour une position v et un état q par :

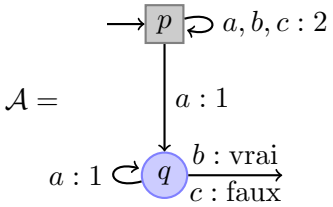
$$\delta_{\mathcal{G}(\mathcal{A},u)}(k,q) = \delta(q, a_k)[r \leftarrow (k+1, r)]$$

avec la convention que $(k, \bot) = \bot$. Un mot u est **accepté** par l'automate $\mathcal{A}$ si Ève gagne le jeu $\mathcal{G}(\mathcal{A}, u)$. L'ensemble des mots acceptés par automate est appelé le **langage accepté par** $\mathcal{A}$ et est noté $L(\mathcal{A})$.

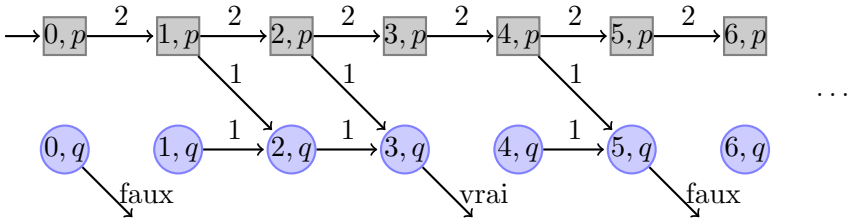
Un automate alternant **préserve la longueur** si $\Delta_0 \subseteq Q \times \mathbb{A}_0 \times \mathbb{C}_0 \times \{\bot\}$. Cela signifie que toute partie du jeu $\mathcal{G}(\mathcal{A}, u)$ pour un automate $\mathcal{A}$ préservant la longueur, a la même longueur que u.

Exemple 7.17. Considérons l'automate de Büchi suivant sur l'alphabet en entrée $\{a, b, c\}$ sans terminateurs :

L'automate est dessiné en utilisant les mêmes conventions que pour les jeux. Les cercles (○) représentent des disjonctions et les carrés (□) représentent des conjonctions. Ainsi, dans ce cas, $\delta(p,a) = (p,a,2,p) \wedge (p,a,1,q)$. Remarquons la possibilité pour l'automate d'arrêter une branche de calcul en utilisant un terminateur. Dans le cas présent $\delta(q,v) = (q,v,\text{vrai},\bot)$ et $\delta(q,c) = (q,c,\text{faux},\bot)$.



L'automate ci-dessus prend en entrée des mots infinis sur l'alphabet $\{a,b,c\}$, et garantit que toute occurrence de la lettre a est suivie d'une occurrence de la lettre b, et qu'aucune occurrence de la lettre c n'est rencontrée avant. Considérons par exemple un mot de la forme $caabac\ldots$, alors le jeu $\mathcal{G}(\mathcal{A}, caabac\ldots)$ est le suivant :



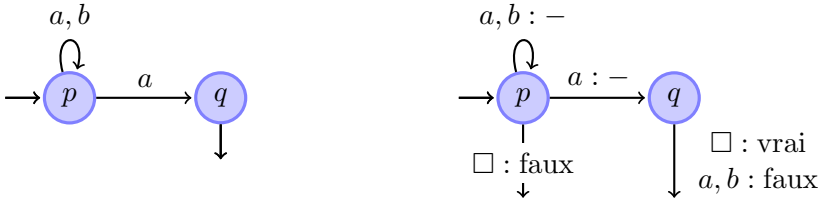
Ce jeu est perdant pour Ève car elle ne peut rien faire pour empêcher Adam d'atteindre la position $5,q$ qui est perdante.

Remarque 7.18 (cas des automates non-déterministes, universels, déterministes et complets). Les automates **non-déterministes** sont ceux pour lesquels

$$\delta(q,a) = \bigvee \Delta(q,a)\ ,$$

pour tout état q et toute lettre a. Cette définition ne coïncide pas syntaxiquement avec les automates non-déterministes de la partie précédente. En effet, les automates de la partie précédente ont la propriété d'accepter ou de rejeter à la fin du mot.

Considérons par exemple l'automate suivant qui accepte tous les mots sur $\{a,b\}$ dont la dernière lettre est un a. L'automate de gauche utilise les notations de la partie précédente.



Le second automate montre le codage en automate alternant, en utilisant la condition de gain d'accessibilité. En particulier, les états finaux sont maintenant codés au moyen de terminateurs (rappelons que $\Box$ représente la fin du mot). Remarquons aussi que de nouvelles transitions sont nécessaires, correspondant au fait que l'automate original n'est pas complet : $(p, a, \text{faux}, \bot)$ et $(p, b, \text{faux}, \bot)$. Cette traduction peut être systématisée. Nous ne rentrons pas davantage dans les détails.

Un automate alternant est **déterministe et complet** s'il est à la fois non-déterministe et universel.

De manière duale, un automate alternant est **universel** si

$$\delta(q, a) = \bigwedge \Delta(q, a)$$

pour tout état q et toute lettre a.

Tout comme pour les jeux, il est naturel de considérer le dual d'un automate alternant. Le **dual** $\overline{\mathcal{A}}$ d'un automate alternant $\mathcal{A}$ s'obtient simplement en remplaçant la condition de gain par son complément, et en dualisant la fonction de transition. Dans le cas de condition de gain borélienne, le résultat de Martin s'applique :

Proposition 7.19. *Pour un automate $\mathcal{A}$ de condition de gain borélienne,*

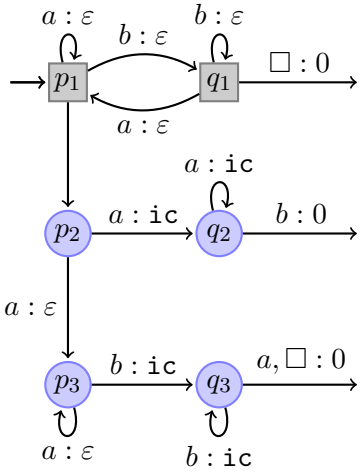
$$L(\mathcal{A}) = L(\overline{\mathcal{A}}) \ .$$

De manière duale, un automate alternant est **universel** si $\delta(q, a)$ est une conjonction pour tout état q et toute lettre a. Un automate alternant est **déterministe et complet** s'il est à la fois non-déterministe et universel.

7.6.2 Automates alternants de coût

Un **automate de coût alternant** $\mathcal{A} = (Q, \mathbb{A}, q_0, O, \delta)$ est défini comme un automate alternant booléen, si ce n'est que la condition de gain est remplacée par un objectif.

Exemple 7.20. Considérons le `distance`-automate alternant suivant.



Pour simplifier les notations, nous avons utilisé une transition non étiquetée entre l'état p_1 et l'état p_2. Elle peut être vue comme une ϵ-transitions, ou comme simplement l'imbrication des connecteurs dans l'expression booléenne $\delta(p_1, a)$. Remarquons que nous n'avons pas décrit toutes les transitions de l'automate. Par défaut, si depuis un état p aucune transition n'est étiquetée x en entrée, alors implicitement, la transition $(p, x, \infty, \bot)$ est présente. C'est le cas dans cet exemple pour les transitions $(p_2, \Box, \infty, \bot)$, $(p_2, \Box, \infty, \bot)$ et $(p_3, \Box, \infty, \bot)$.

Cet automate, étant donné un mot en entrée u renvoie $\max_i \min(m_i, n_i)$ si u s'écrit $a^{m_1} b_{n_1} \dots a^{m_k} b^{n_k}$ avec $m_i \geq 1$ et $n_i \geq 1$ pour tout $i = 1 \dots k$, et ∞ sinon.

Nous avons présenté au cours de la section précédente comment, à partir d'un automate alternant $\mathcal{A}$ et d'un mot u, construire un jeu $\mathcal{G}(\mathcal{A}, u)$. La même construction s'utilise dans le cadre des automates alternants de coût, à la différence que cette fois-ci le jeu produit est de coût. Nous définissons la sémantique d'un automate alternant $[\![\mathcal{A}]\!]$ tout simplement par :

$$[\![\mathcal{A}]\!](u) = \mathit{valeur}(\mathcal{G}(\mathcal{A}, u)) \ ,$$

pour tout mot u sur l'alphabet d'entrée.

Les automates **préservant la longueur**, les automates **non-déterministes**, les automates **universels** et les automates **déterministes et complets** se définissent, comme dans le cas booléen, au moyen d'une restriction des fonctions de transition possibles. La notion de **dual** d'un automate est également identique (il faut bien entendu remplacer l'objectif par son dual).

Nous nommerons **`B`-automate alternant**, (*resp.* **`S`-automate alternant**, *resp.* **`S`-automate alternant** ou **`hB`-automate alternant**) les automates alternants à coût utilisant un objectif `B` (*resp.* un objectif `S`, *resp.* un objectif `hB`). Cette terminologie s'étend naturellement à des conditions plus complexes comme ($\texttt{hB} \wedge \texttt{Büchi}$), etc...

De la proposition 7.9 (*c.-à-d.* le théorème de Martin) nous déduisons directement :

Proposition 7.21. *Pour un automate $\mathcal{A}$ d'objectif borélien,*

$$[\![\mathcal{A}]\!] = [\![\overline{\mathcal{A}}]\!] \ .$$

7.7 Automates déterministes en histoire

Nous présentons dans ce chapitre la notion d'automate déterministe en histoire. Il s'agit d'automates alternants (ou le plus souvent, non déterministes) qui possèdent certaines caractéristiques les rapprochant des automates déterministes. En particulier, nous verrons au cours de la section suivante (section 7.8) que de tels automates «se composent bien» entre eux, et en particulier avec les jeux.

Les automates déterministes en histoire ont été introduits en premier par Henzinger et Piterman dans le cas des automates non-déterministes booléens [47] sous le nom d'automates «bons pour résoudre les jeux» (terminologie faisant référence aux bonnes caractéristiques de composition qu'ils possèdent). La notion a été découverte indépendamment dans le cadre des fonctions de coût ([25] et [33]). Cette thèse est la première occasion de présenter la notion de déterminisme en histoire dans toute sa généralité, c'est à dire pour les automates alternants en soulignant les liens avec la théorie des jeux.

Là encore, nous présentons tout d'abord la notion dans le cadre booléen (section 7.7.1) pour ensuite l'adapter aux fonctions de coût (section 7.7.2)

7.7.1 Déterminisme en histoire pour les automates booléens

Il s'agit de définir une notion de stratégie pour les automates alternants. Fixons nous donc un automate alternant booléen $\mathcal{A} = (Q, \mathbb{A}, q_0, W, \delta)$. Cette notion étend celle de stratégie dans un jeu, et il est donc bon de garder en tête la définition d'une stratégie dans un jeu (page 106).

Une **stratégie de traduction pour Ève** σ_E (ou plus simplement une **stratégie**) dans l'automate $\mathcal{A}$ est un ensemble de chemins dans $\mathcal{A}$ d'origine q_0 décrits comme des mots sur (Δ_0, Δ_1) (vu comme un alphabet avec terminateur) et tel que :

- σ_E est topologiquement clos,
- pour tout chemin partiel $\pi \in \mathit{pref}(\sigma_E)$ se terminant en $q \in Q$ et pour toute lettre $a \in \mathbb{A}$

$$\bigwedge \sigma_{\mathrm{E}}[\pi, a] \qquad \Rightarrow \qquad \delta(q, a) \ , \tag{7.2}$$

  où $\sigma_{\mathrm{E}}[\pi, a] = \{(t \in \Delta(q, a) \ : \ \pi t \in \mathit{pref}(\sigma_E)\}$.

Il est assez clair que cette définition coïncide avec la notion de stratégie dans les jeux dans le cas de l'alphabet entrée $\mathbb{I}$. Il s'agit en fait d'une sorte de «stratégie générique», que l'on peut utiliser sur les jeux de la forme $\mathcal{G}(\mathcal{A}, u)$.

Étant donnés une stratégie de traduction pour Ève σ_E et un mot u sur l'alphabet d'entrée, la **stratégie induite** par σ_E sur u est :

$$\sigma_{\mathrm{E}}^u = \{(p_0, c_1, p_1)(p_1, c_2, p_2)\dots \ : \\ (p_0, a_1, c_1, p_1)(p_1, a_2, c_2, p_2)\dots \in \sigma_E, \quad u = a_1 a_2 \dots\} \ .$$

Il s'agit d'une stratégie pour Ève dans le jeu $\mathcal{G}(\mathcal{A}, u)$. De manière duale, si σ_A est une stratégie de traduction pour Adam, et u est un mot en entrée de l'automate, alors il existe une stratégie induite par σ_{A} sur u, notée σ_{A}^u qui est une stratégie pour Adam dans le jeu de $\mathcal{G}(\mathcal{A}, u)$.

Un automate alternant booléen $\mathcal{A}$ de condition d'acceptation W est dit **déterministe en histoire** s'il existe deux stratégies de traduction σ_{E} et σ_{A}, pour Ève et pour Adam respectivement, telles que pour tout mot u, l'une des deux situations suivantes est satisfaite :

- soit σ_{E}^u est une stratégie gagnante pour Ève dans le jeu $\mathcal{G}(\mathcal{A}, u)$,
- soit σ_{A}^u est une stratégie gagnante pour Adam dans le jeu $\mathcal{G}(\mathcal{A}, u)$.

Remarquons que cette propriété implique que $\mathcal{G}(\mathcal{A}, u)$ est déterminé pour tout u. Il est assez évident que si un automate alternant est déterministe en histoire, alors son dual l'est aussi. Il suffit pour cela d'échanger les stratégies de traduction d'Ève et d'Adam.

En pratique, dans les preuves, nous établissons la propriété équivalente suivante :

- pour toute partie $(p_0, a_1, c_1, p_1)\cdots \in \sigma_{\mathrm{E}}$, si $a_1 a_2 \dots$ est accepté par $\mathcal{A}$, alors $c_1 c_2 \cdots \in W$,
- pour toute partie $(p_0, a_1, c_1, p_1)\cdots \in \sigma_{\mathrm{A}}$, si $a_1 a_2 \dots$ n'est pas accepté par $\mathcal{A}$, alors $c_1 c_2 \cdots \notin W$.

Remarque 7.22 (cas des jeux). Dans le cas des jeux, c'est à dire le cas où l'alphabet en entrée est $\mathbb{I}$ (voir la remarque 7.16), la notion de stratégie de traduction coïncide avec la notion de stratégie, et un jeu, vu comme un automate alternant sur l'alphabet $\mathbb{I}$, est déterministe en histoire si et seulement si le jeu est déterminé. Ainsi, une façon de comprendre la notion de déterminisme en histoire est comme un extension aux automates de la notion de détermination dans les jeux.

Dans ce document, la notion de déterminisme en histoire est le plus souvent utilisée pour les automates non-déterministes. La remarque suivante détaille ce point.

Remarque 7.23 (cas des automates non-déterministes ou universels). Remarquons que si l'automate est universel, alors l'implication (7.2) revient à énoncer

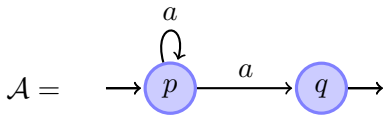
$$\bigwedge \sigma_{\mathrm{E}}[\pi, a] \qquad \Rightarrow \qquad \bigwedge \Delta(q, a) \ ,$$

ce qui équivaut à $\sigma_{\mathrm{E}}[\pi, a] = \Delta(\pi, a)$. Pour cette raison, il n'existe qu'une et une seule stratégie de traduction pour Ève, celle qui contient tous les chemins de l'automate. (Par analogie, dans un jeu dans lequel Ève ne joue jamais, alors il n'existe qu'une stratégie pour Ève, puisqu'elle n'a jamais l'occasion de jouer.) De manière duale, si l'automate est non-déterministe, alors il n'existe qu'une et une seule stratégie de traduction pour Adam.

Pour cette raison, dans le cas d'un automate non-déterministe, la notion de déterminisme en histoire peut être simplifiée. Un automate non-déterministe est déterministe en histoire s'il existe une stratégie de traduction σ_{E} pour Ève telle que pour tout mot u accepté par l'automate, σ_{E}^{u} est une stratégie gagnante pour Ève dans le jeu $\mathcal{G}(\mathcal{A}, u)$.

Exemple 7.24. Tous les automates déterministes sont déterministes en histoire. Dans ce cas la stratégie est simplement l'ensemble des chemins issus de l'état initial dans l'automate. Nous verrons que dans le cas des langages réguliers, il n'existe pas vraiment d'autres exemples d'automates déterministes en histoire (théorème 7.25).

L'automate suivant n'est pas déterministe en histoire.



En effet, cet automate accepte tous les mots (sur la lettre a), sauf le mot vide $\square$. En revanche, il n'existe aucune stratégies de traduction σ_{E} qui le rende déterministe en histoire. Pour raisonner correctement, listons les transitions de cet automate :

$$(p, a, -, p),\ (p, a, -, q), (q, a, \text{faux}, \bot), (p, \square, \text{faux}, \bot), (q, \square, \text{vrai}, \bot)$$

L'automate accepte le mot $a\square$. Cela signifie que Ève gagne le jeu $\mathcal{G}(\mathcal{A}, a\square)$. La seule façon pour Ève de gagner ce jeu est de jouer $(p, -, q)(q, \square, \bot)$. Donc nécessairement $\sigma_{\mathrm{E}}^{a\square} = \{(p, a, q)(q, \text{vrai}, \bot)\}$. Par conséquent, nécessairement, $(p, a, -, q)(q, \square, \text{vrai}, \bot) \in \sigma_{\mathrm{E}}$, et donc $(p, a, -, q) \in \mathit{pref}(\sigma_{\mathrm{E}})$. L'implication 7.2 entraîne alors que $(p, a, -, q)(q, a, \text{faux}, \bot) \in \sigma_{\mathrm{E}}$. Par conséquent, $\sigma_{\mathrm{E}}^{aa\square}$ ne gagne pas le jeu $\mathcal{G}(\mathcal{A}, aa\square)$. Cela contredit le déterminisme en histoire puisque $aa\square$ est accepté par l'automate.

Nous verrons que cela pouvait s'obtenir directement au moyen du théorème 7.25 qui énonce qu'un automate non-déterministe de mot fini est déterministe en histoire si et seulement si il peut être rendu déterministe (tout en acceptant le même langage) en retirant des transitions. Bien entendu, l'automate ci-dessus ne peut pas être déterminisé de la sorte.

Une façon d'appréhender la distinction entre le déterminisme usuel et le déterminisme en histoire, est de considérer que le déterminisme standard est un déterminisme «syntaxique», c'est à dire que la stratégie à appliquer est claire puisqu'il n'existe qu'une et une seule transition possible pour chaque paire (état,lettre). La situation est différente dans le cas du déterminisme en histoire, pour lequel le choix de la transition est donné à la stratégie. La stratégie peut faire référence au passé de l'exécution (l'histoire) pour faire son choix, alors que le déterminisme usuel détermine la transition à appliquer uniquement en fonction de l'état courant, qui est une information «syntaxique».

Dans le cas des automates non-déterministes de mot fini, les automates déterministes en histoire sont bien compris.

Théorème 7.25 (C. [26] et [67] pour l'algorithme)**.** *Un automate fini non-déterministe booléen de mot fini est déterministe en histoire si et seulement s'il peut être rendu déterministe en retirant des transitions et en ne gardant qu'un état initial, tout en acceptant toujours le même langage.*

Cette propriété se teste en temps polynomial et l'automate déterministe se produit en temps polynomial.

La propriété n'est plus vraie pour les automates de mots infinis.

Proposition 7.26 (Boker [11])**.** *Il existe un automate non-déterministe de Büchi déterministe en histoire qui ne peut être rendu déterministe par élimination de transitions.*

La conjecture dans le cas des mots infinis est la suivante.

Conjecture 7.27. *Pour tout automate déterministe en histoire de parité sur les mots infinis, il existe un automate de parité déterministe ayant au plus le même nombre d'états, et acceptant le même langage.*

Ce que signifie le résultat ci-dessus, ainsi que la conjecture, est qu'il n'y a pas grand chose à gagner à raffiner la notion de déterminisme et la remplacer par la notion de déterminisme en histoire dans la théorie des langages réguliers.

Nous allons voir au cours du chapitre suivant que, dans le contexte des fonctions de coût, la situation change radicalement, et la notion de déterminisme en histoire devient incontournable.

7.7.2 Déterminisme en histoire pour les automates à coût

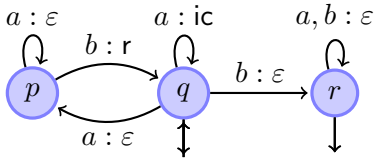
La notion de déterminisme en histoire s'adapte assez naturellement au cas des automates à coût. La seule subtilité est qu'une fonction de correction doit être ajoutée (tout comme pour la définition de l'$\approx$-mémoire) et qu'une stratégie de traduction doit être donnée pour chaque n.

Un automate alternant $\mathcal{A} = (Q, \mathbb{A}, q_0, O, \delta)$ d'objectif $O = (\mathbb{C}, f, \min)$ (le cas max s'obtient par dualité) est **α-déterministe en histoire** pour une fonction de correction α si pour tout entier n il existe une paire de stratégie de traduction σ_E et σ_A pour Ève et Adam respectivement, telles que pour tout mot u,

– soit la stratégie d'Ève σ_{E}^u est gagnante dans le jeu $\mathcal{G}(\mathcal{A}, u)[O_{\alpha(n)}]$.
– soit la stratégie d'Adam σ_{A}^u est gagnante dans le jeu $\mathcal{G}(\mathcal{A}, u)[O_n]$.

Remarquons que, contrairement au cas booléen, les deux cas peuvent être vrai simultanément pour un même mot u.

Exemple 7.28. L'automate suivant est un B-automate à 1 compteur, déterministe en histoire, qui accepte la fonction minseg de l'exemple 5.6, c'est à dire la fonction qui à $a^{n_0}ba^{n_1}\dots ba^{n_k}$ associe $\min(n_1, \dots, n_k)$:



Montrer que cet automate est bien déterministe en histoire (dans le cas présent pour $\alpha = Id$) et accepte bien la fonction minseg se fait en deux étapes. Il s'agit (a) de montrer que si cet automate n-accepte un mot u, alors $\text{minseg}(u) \leq n$, et (b) montrer que pour tout entier n, il existe une stratégie de traduction telle que pour tout mot u tel que $\text{minseg}(u) \leq n$, u est accepté par l'automate en suivant la stratégie de traduction.

(a) Supposons qu'il existe une n-exécution sur un mot u. Cela signifie que la valeur du compteur ne dépasse jamais n. Ainsi, l'automate était nécessairement dans l'état q avec pour valeur du compteur 0 (*c.-à-d.* après un b ou le début du mot) puis a lu une séquence d'au plus n occurrences consécutives de la lettre a, suivie d'une lettre b, ou de la fin du mot. Il s'ensuit que $\text{minseg}(u) \leq n$.

(b) Il s'agit de décrire la stratégie de traduction pour une valeur de n fixée. Le seul cas où il peut y avoir ambiguïté est le cas de l'automate dans l'état q, lisant la lettre a. En effet, dans ce cas, l'automate doit choisir entre aller en p (appelons ce choix «abandonner») ou rester en q, en incrémentant le compteur (appelons ce choix «continuer»). Considérons la stratégie de traduction δ_n telle que dans l'état q, en lisant la lettre a en entrée :

– si la valeur du compteur est au plus $n-1$, alors δ_n choisit la transition «continue»,
– sinon (si la valeur est n), alors δ_n choisit la transition «abandonne».

Remarquons tout d'abord que, en suivant cette stratégie, le compteur ne peut jamais dépasser la valeur n.

Considérons maintenant un mot u tel que $\text{minseg}(u) \leq n$. Remarquons aussi que si l'automate commence, en suivant δ_n, à lire un segment de a dans l'état q, alors le compteur à cet instant possède une valeur nulle. Si de plus ce segment a une longueur inférieure ou égale à n, alors l'automate est toujours dans l'état q à la fin de ce segment (en suivant δ_n !). Ainsi, à la fin du premier segment de a de longueur au plus n apparaissant dans le mot, l'automate se trouve dans l'état q. Si le mot se termine à ce moment, il est accepté, car q est final. Sinon, si le mot continue par un b, alors l'automate passe dans l'état r, et accepte inéluctablement le mot u car r est final également.

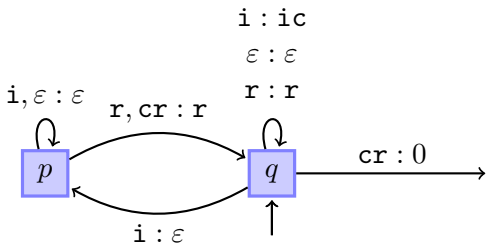
Nous aurons l'occasion d'utiliser au cours de ce travail plusieurs automates déterministes en histoire, pour des raisons différentes. Nous allons présenter ces résultats dans la suite de cette section. La proposition 7.29 met en relation les objectifs `S` et `¬B`. Elle sera utilisée en particulier toutes les fois que la combinatoire de la condition `S` sera problématique. Nous verrons ensuite la proposition 7.30 qui permet de passer de l'objectif `B` à l'objectif `hB`.

La proposition suivante nous permet de passer de l'objectif `S` à l'objectif `¬B` et vis-versa. En particulier, c'est grâce à ce résultat que nous ne chercherons jamais à résoudre

directement les jeux impliquant des conditions S, puisque nous avons la possibilité de les transformer en jeux n'utilisant que la condition ¬B, qui eux possèdent de meilleures propriétés.

Proposition 7.29. *Il existe un* ¬B*-automate universel déterministe en histoire pour l'objectif* S*. Il existe un* S*-automate non-déterministe déterministe en histoire pour l'objectif* ¬B*.*

Démonstration. Nous n'effectuons que la première de ces réductions (la seule qui nous intéresse pou la suite), et seulement pour un unique compteur. Pour un nombre plus grand de compteurs, il suffit d'effectuer une telle construction de manière concurrente pour chaque compteur (voir aussi section 7.8.3 pour traiter de l'exécution concurrente de plusieurs conditions).



Dans cette description, nous avons omis les terminateurs. Ainsi, de tout état, une transition 0 : 0 est possible, ainsi, qu'une transition ∞ : ∞. Le fonctionnement de cet automate est très similaire à celui de l'exemple 7.28. CQFD

Une autre remarque importante est que tout automate déterministe est déterministe en histoire. En fait, nous avons déjà vu dans la partie sur les mots un automate déterministe important : «l'enregistrement de dernière apparition». Ainsi, la proposition suivante est tous simplement une présentation différente du théorème 5.19.

Proposition 7.30. *Il existe un* hB*-automate déterministe pour* B*. Les deux objectifs utilisent le même nombre de compteurs. Il existe un* ¬hB*-automate déterministe pour* ¬B*. Les deux objectifs utilisent le même nombre de compteurs.*

Remarquons qu'il s'agit en fait deux fois du même automate, et seule la condition d'acceptation change. Cette proposition nous permet de passer directement d'une condition B à une condition hB, et d'une condition ¬B à une condition ¬hB. Comme les conditions hiérarchisées ont de meilleures propriétés de mémoire, cela permet, là encore, de simplifier certains problèmes.

La proposition précédente illustrait à quel point il faut être attentif à la différence entre B et ¬B. Or, nous avons vu que, essentiellement, $\overline{\mathtt{B}}$ équivaut à ¬B ∧ accessibilité, ainsi que $\overline{\neg \mathtt{B}}$ équivaut à B ∧ accessibilité. Comme les conditions de Büchi et de co-Büchi permettent d'exprimer les conditions d'accessibilité, nous en tirons la proposition suivante.

Proposition 7.31. *Il existe un* $\neg\mathtt{B} \wedge \mathtt{Büchi}$*-automate non-déterministe pour* $\overline{\mathtt{B}} \vee \mathtt{Büchi}$*. Il existe un* $\mathtt{B} \wedge \mathtt{Büchi}$*-automate non-déterministe pour* $\overline{\neg\mathtt{B}} \vee \mathtt{Büchi}$*.*

Nous avons rencontré la proposition 5.16 qui énonçait que les fonctions régulières de coût sur les mots finis n'étaient pas toutes acceptées par des B-automates déterministes. Ce point négatif est contrebalancé par le fait que toutes les fonctions de coût régulières sont acceptées par des automates à coût déterministes en histoire.

Théorème 7.32 (Theorème 1 dans [25])**.** *Pour toute fonction régulière de coût* f *sur les mots finis, il existe effectivement un* B*-automate non-déterministe déterministe en histoire pour* f *et un* S*-automate non-déterministe déterministe en histoire pour* f*.*

Il est possible d'en déduire un résultat légèrement plus général pour les conditions de sûreté, par passage à la limite.

Corollaire 7.33. *Si* f *est une fonction régulière de coût sur les mots finis, alors*
- *la fonction qui à chaque mot infini* v *associe*

$$f^{\sup}(v) = \sup\{f(u) \ : \ u \text{ préfixe fini de } v\} \ ,$$

  est acceptée par un hB*-automate non-déterministe, déterministe en histoire.*
- *la fonction qui à chaque mot infini associe*

$$f^{\inf}(v) = \inf\{f(u) \ : \ u \text{ préfixe fini de } v\} \ ,$$

  est acceptée par un S*-automate non-déterministe, déterministe en histoire, et également par un* $\neg$B*-automate universel déterministe en histoire*

Il s'agit d'une construction élémentaire : l'automate pour f est vu comme un automate de mot fini, qui doit rester dans un état acceptant/rejetant (sinon, l'exécution s'arrête). Il est aisé de vérifier que cette construction est correcte.

Remarque 7.34. On pourrait se demander si, par exemple, $f^{\inf}$ peut être accepté par un B-automate déterministe en histoire. La réponse et non, et ce indépendamment du fait que l'automate soit alternant ou non-déterministe, et indépendamment du fait qu'il soit déterministe en histoire. Le problème, est que le B-automate doit être capable de trouver un préfixe témoin du fait que $f^{\inf}(v) \leq n$. Or, la condition $\mathtt{B}_n$ est par nature topologiquement close, *c.-à-d.* qu'elle ne peut que vérifier que tous les préfixes satisfont une propriété. En revanche elle ne permet pas de garantir qu'il existe un préfixe satisfaisant un propriété donnée.

Précisons ce contre-exemple. Considérons un B-automate sur les mots infinis acceptant une fonction g et un entier k, il est aisé de construire un automate de sûreté (dont toutes les exécutions sont acceptantes) qui reconnait le langage

$$L_g = \{v \ : \ g(v) \leq k\} \ .$$

En effet, il suffit de maintenir dans les états de l'automate la valeur du compteur, jusqu'à k.

Considérons maintenant la fonction f qui à un mot fini u associe 0 si u contient la lettre a, et ∞ sinon. La fonction $f^{\inf}$ associe alors à un mot infini la valeur 0 s'il contient un a, et la valeur ∞ sinon. Supposons qu'un $\mathtt{B}$-automate accepte un fonction g qui soit $\approx$-équivalent à $f^{\inf}$. Cela signifie qu'il existe un entier k tel que si $f^{\inf}(v) = 0$, $g(v) \leq k$, et sinon, $g(v) = \infty$. Ainsi, le langage L_g ci-dessus contiendra exactement les mots possédant une occurrence de la lettre a. Pourtant, il est bien connu qu'il n'existe pas d'automate de sûreté pour ce langage.

Ainsi, cette impossibilité est entièrement héritée de la théorie classique.

En fait, en utilisant des techniques similaires au cas des mots finis, il est possible d'établir le résultat plus général suivant.

Théorème 7.35 (non-publié)**.** *Étant donné une fonction de coût régulière sur les mots infinis f, alors*

- *f est effectivement acceptée par un $\mathtt{B} \wedge \mathtt{parité}$-automate non-déterministe, déterministe en histoire,*
- *et f est effectivement acceptée par un $\mathtt{S} \wedge \mathtt{parité}$-automate non-déterministe, déterministe en histoire.*

7.8 La composition d'automates alternants

La motivation première pour introduire les automates déterministes en histoire est la possibilité de résoudre des jeux. C'est la raison pour laquelle ces automates ont été introduits dans le cas des langages sous le nom d'automates «bons pour résoudre les jeux» [47]. Au cours de cette section, nous développons cet aspect. En fait, la présentation donnée ici est plus générale puisqu'elle consiste à unifier les automates et les jeux, et par la même à décrire des constructions générales qui, comme cas particulier, permettent de résoudre les jeux.

Comme dans les section précédentes, nous commençons par présenter l'approche dans le cas booléens (section 7.8.1) puis dans le cas des fonctions de coût (section 7.8.2).

7.8.1 Composition d'automates booléens

Principe de la composition. Il est possible de définir une notion de composition sur les automates. Un automate alternant $\mathcal{A}$ peut en effet être vu comme une machine réduisant l'acceptation du mot en entrée à une combinaison booléenne positive de questions concernant la condition d'acceptation : la disjonction sur toutes les stratégies possibles de Ève dans le jeu $\mathcal{G}(\mathcal{A}, u)$ de la conjonction sur toutes les parties qu'elle contient, de propriétés de la forme «la partie est gagnante pour Ève». Vu comme cela, composer des automates consiste simplement à effectuer une substitution de formules, capturant la composition des réductions.

Composition sémantique. Notre objectif est (en version booléenne) de transformer un automate alternant $\mathcal{A}$ pour le langage L utilisant la condition de gain K et un automate alternant $\mathcal{B}$ pour le langage K utilisant la condition de gain W en un automate alternant pour le langage L utilisant la condition de gain W.

Composition syntaxique. La particularité des automates est qu'ils ne peuvent construire cette combinaison booléenne que progressivement, au fur et à mesure que le mot en entrée est consommé. Pour cette raison, il est impossible d'implémenter l'opération de composition en général. Il existe néanmoins une façon naturelle de composer syntaxiquement les automates alternants, que nous décrivons dans la suite de cette section. Cette composition syntaxique est incorrecte en général. Nous verrons que la notion de déterminisme en histoire réconcilie la notions de «composition syntaxique» avec celle de «composition sémantique».

Une approche naturelle consiste à construire un automate produit qui exécute l'automate $\mathcal{A}$, puis, à chaque fois qu'une lettre de l'alphabet de S est produite, exécute immédiatement une transition de $\mathcal{B}$ correspondante. Cette construction est en général incorrecte (en ce sens qu'elle n'effectue pas la composition sémantique des automates). En fait elle est correcte si $\mathcal{A}$ ou $\mathcal{B}$ est déterministe. Elle est aussi correcte si les deux automates sont non-déterministes (ou tous deux universels). Nous verrons que si l'automate $\mathcal{B}$ est déterministe en histoire, cette comstruction est encore correcte.

Considérons deux automates alternants à coût $\mathcal{A} = (Q_\mathcal{A}, \mathbb{A}, q_0, \delta_\mathcal{A}, W_\mathcal{A})$ et $\mathcal{B} = (Q_\mathcal{B}, \mathbb{B}, q_0, \delta_\mathcal{B}, W_\mathcal{B})$ tels que $\mathcal{B}$ accepte le langage $W_\mathcal{A}$. L'automate $\mathcal{A} \triangleleft \mathcal{B}$ obtenu par **composition** (syntaxique) de $\mathcal{A}$ avec $\mathcal{B}$ est tel que :

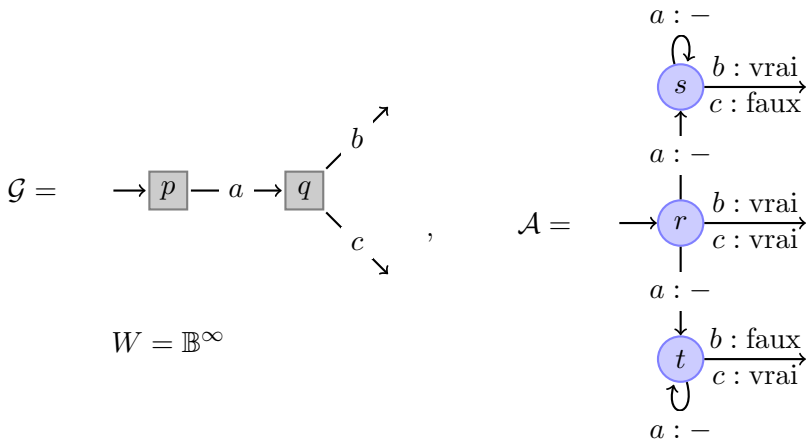
- son alphabet d'entrée est $\mathbb{A}$ et sa condition d'acceptation $W_\mathcal{B}$,
- l'ensemble d'états est $Q_\mathcal{A} \times Q_\mathcal{B}$,
- pour tous $p \in Q_\mathbb{A}$, $q \in Q_\mathbb{B}$ et $a \in \mathbb{A}$, $\delta((p,q),a)$ se définit comme :

$$\delta_\mathcal{A}(p,a) \left[\begin{array}{l} (p,a,b,p') \leftarrow \\ \qquad \delta_\mathcal{B}(q,b)[(q,b,c,q') \leftarrow ((p,q),a,c,(p',q'))] \end{array} \right] . \tag{7.3}$$

Remarquons dans cette expression que, si a ou b sont des terminateurs, alors c l'est nécessairement aussi, et dans ce cas q' est $\bot$. Nous prenons comme convention que $(p', \bot) = \bot$.

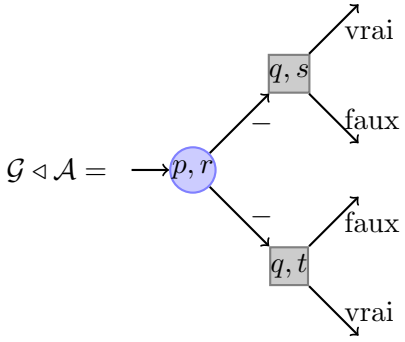
En gardant en mémoire que tout jeu peut être vu comme un automate alternant (remarque 7.16), il est naturel d'appliquer cette construction produit aussi pour composer un jeu avec un automate.

Exemple 7.36. L'exemple consiste à composer un jeu $\mathcal{G}$ (*c.-à-d.* un automate avec l'alphabet $\mathbb{I}$ en entrée) sur un alphabet $\mathbb{B}$ contenant une lettre a et deux terminateurs b, c avec un automate alternant sur les mots finis :



Le jeu ne possède que deux parties $(p, a, q)(q, b, \bot)$ et $(p, a, q)(q, c, \bot)$ et la condition d'acceptation est que tous les mots sont gagnants. L'automate accepte bien tous les mots de $\mathbb{B}^\infty$. Une composition sémantique correcte devrait donc produire un jeu gagné par Ève.

Pourtant, le résultat du produit syntaxique du jeu avec l'automate est le suivant :



Ce jeu est bien évidemment perdant pour Ève, et ne correspond donc pas à la composition sémantique. La raison de ce problème est évidente. L'automate $\mathcal{A}$ est non-déterministe, et lors de la première transition par a, il devine si le terminateur du mot en entrée est b ou c. Or dans le jeu $\mathcal{G}$, Adam a la possibilité d'attendre un coup, en produisant la lettre a, ce qui force Ève, dans le jeu $\mathcal{G} \triangleleft \mathcal{A}$, à deviner si le terminateur sera b ou c. Adam a alors la possibilité de choisir de produire b ou c de telle sorte que le choix d'Ève s'avère incorrect. En d'autres termes, Ève est forcée de faire un choix, qu'Adam connaît, et dont il tire parti pour gagner le jeu.

Comme mentionné précédemment, la raison de l'introduction de la notion de déterminisme en histoire est que les automates déterministes en histoire se composent bien avec les jeux (contrairement aux automates généraux comme le montre l'exemple précédent). La

proposition suivante est plus générale puisqu'elle fait référence à la composition d'automates alternants.

Proposition 7.37. *Si $\mathcal{A}$ et $\mathcal{B}$ sont deux automates alternants, tels que le langage de $\mathcal{B}$ coïncide avec la condition d'acceptation de $\mathcal{A}$, alors :*
- *si $\mathcal{B}$ est déterministe en histoire, alors $\mathcal{A}$ et $\mathcal{A} \triangleleft \mathcal{B}$ acceptent le même langage,*
- *si $\mathcal{A}$ et $\mathcal{B}$ sont tous deux déterministes en histoire, alors $\mathcal{A} \triangleleft \mathcal{B}$ est déterministe en histoire.*

Démonstration. Fixons dans cette preuve deux automates alternants :

$$\mathcal{A} = (Q_{\mathcal{A}}, \mathbb{A}, q_0^{\mathcal{A}}, W_{\mathcal{A}}, \delta_{\mathcal{A}}) \qquad \text{et} \quad \mathcal{A} = (Q_{\mathcal{B}}, \mathbb{B}, q_0^{\mathcal{B}}, W_{\mathcal{B}}, \delta_{\mathcal{B}}) \ ,$$

tels que le langage accepté par $\mathcal{B}$ est $W_{\mathcal{A}}$. Le cœur de la preuve consiste à décrire comment composer les stratégies de traduction dans $\mathcal{A}$ et dans $\mathcal{B}$ pour produire des stratégies de traduction dans $\mathcal{A} \triangleleft \mathcal{B}$. La preuve commence par décrire cette opération de composition, et à montrer que cela produit bien une stratégie de traduction dans $\mathcal{A} \triangleleft \mathcal{B}$. Elle continue en établissant le résultat pour la composition d'automates déterministes en histoire, puis pour la composition d'un jeu avec un automate déterministe en histoire, puis finalement pour la composition d'un automate avec un automate déterministe en histoire.

Composition de stratégies de traduction. Soit σ_E une stratégie de traduction pour Ève dans $\mathcal{A}$ et τ_E une stratégie de traduction pour Ève dans $\mathcal{B}$, la **stratégie composée** $\sigma_E \triangleleft \tau_E$ est :

$$\begin{aligned}\sigma_E \triangleleft \tau_E = \{((p_0,q_0),a_1,c_1,(p_1,q_1))((p_1,q_1),a_2,c_2,(p_2,q_2))\dots \ : \\ (p_0,a_1,b_1,p_1)(p_1,a_2,b_2,p_2)\cdots \in \sigma_E, \\ (q_0,b_1,c_1,q_1)(q_1,b_2,c_2,q_2)\cdots \in \tau_E\}\end{aligned}$$

Prouvons tout d'abord qu'il s'agit bien d'une stratégie de traduction pour Ève dans $\mathcal{A} \triangleleft \mathcal{B}$. Il est évident que $\sigma_E \triangleleft \tau_E$ est topologiquement clos. Il s'agit maintenant d'établir l'implication (7.2) pour toute partie partielle $\pi'' \in \mathit{pref}(\sigma_E \triangleleft \tau_E)$ se terminant en (p,q) et toute lettre a. Remarquons tout d'abord que par construction de $\sigma_E \triangleleft \tau_E$, il existe

$$\begin{aligned}\pi &= (p_0,a_1,b_1,p_1)\dots(p_{k-1},a_k,b_k,p_k)\cdots \in \mathit{pref}(\sigma_E) \ , \\ \text{et} \quad \pi' &= (q_0,b_1,c_1,q_1)\dots(q_{k-1},b_k,c_k,q_k)\cdots \in \mathit{pref}(\tau_E) \ ,\end{aligned}$$

(avec $p = p_k$ et $q = q_k$) telles que

$$\pi'' = ((p_0,q_0),a_1,c_1,(p_1,q_1))\dots((p_{k-1},q_{k-1}),a_k,c_k,(p_k,q_k)) \ .$$

Nous obtenons donc, par construction de $\sigma_E \triangleleft \tau_E$:

$$(\sigma_E \triangleleft \tau_E)[\pi'',a] = \bigcup_{(p,a,b,p')\in\sigma_E[\pi,a]} \ \bigcup_{(q,b,c,q')\in\tau_E[\pi',b]} \{((p,q),a,c,(p',q'))\} \ ,$$

ce qui se traduit directement en termes logiques par l'équivalence entre $\bigwedge((\sigma_E \triangleleft \tau_E)[\pi'', a])$ et :

$$\left(\bigwedge \sigma_E[\pi, a]\right)\left[\begin{array}{l}(p, a, b, p') \leftarrow \\ \quad (\bigwedge \tau_E[\pi', b])\,[(q, b, c, q') \leftarrow ((p, q), a, c, (p', q'))]\end{array}\right]$$

Remarquons ici qu'il s'agit exactement de la même expression que pour la définition de $\delta((p, q), a)$ dans la composition syntaxique des automates (7.3), dans laquelle $\delta_{\mathcal{A}}(p, a)$ est remplacé par $\bigwedge \sigma_E[\pi, a]$ et $\delta_{\mathcal{B}}(q, b)$ par $\bigwedge \tau_E[\pi', b]$. Comme par hypothèse σ_E et τ_E sont des stratégies de traduction, nous avons

$$\bigwedge \sigma_E[\pi, a] \Rightarrow \delta_{\mathcal{A}}(p, a) \qquad \text{et} \quad \bigwedge \tau_E[\pi', b] \Rightarrow \delta_{\mathcal{B}}(q, b)\ .$$

Comme de plus les formules ne contiennent pas de négations, il s'ensuit que $\bigwedge(\sigma_E \triangleleft \tau_E)[\pi'', a] \Rightarrow \delta((p, q), a)$, ce qui établit l'implication (7.2). $\sigma_E \triangleleft \tau_E$ est donc bien une stratégie de traduction sur l'automate $\mathcal{A} \triangleleft \mathcal{B}$.

Composition d'un déterministe en histoire avec un déterministe en histoire. Intéressons nous maintenant au cas de $\mathcal{A}$ et $\mathcal{B}$ déterministes en histoire. Soit σ_E, σ_A et τ_E, τ_A les stratégies de traduction correspondantes. Il s'agit de montrer que $\sigma_E \triangleleft \tau_E, \sigma_A \triangleleft \tau_A$ témoignent bien du déterminisme en histoire de $\mathcal{A} \triangleleft \mathcal{B}$. Considérons donc un mot $u = a_1 \dots$, et supposons qu'il est accepté par $\mathcal{A}$. Considérons maintenant une partie

$$((p_0, q_0), a_1, c_1, (p_1, q_1)) \cdots \in (\sigma_E \triangleleft \tau_E)\ .$$

Il s'agit de montrer que $c_1 c_2 \cdots \in W_{\mathcal{B}}$. Pour cela, remarquons tout d'abord qu'il existe $b_1 b_2 \dots$ tel que (a) $(p_0, a_1, b_1, p_1) \cdots \in \sigma_E$, et (b) $(q_0, b_1, c_1, q_1) \cdots \in \tau_E$. Comme σ_E témoigne du déterminisme en histoire, nous déduisons de (a) que $b_1 b_2 \cdots \in W_{\mathcal{A}}$. Cela signifie par hypothèse que $b_1 b_2 \dots$ est accepté par $\mathcal{B}$. Comme cette fois-ci τ_E témoigne du déterminisme en histoire de $\mathcal{B}$, nous obtenons de (b) que $c_1 c_2 \cdots \in W_{\mathcal{B}}$. Ainsi, $(\sigma_E \triangleleft \tau_E)^u$ est une stratégie gagnante pour Ève dans le jeu $\mathcal{G}(\mathcal{A} \triangleleft \mathcal{B}, u)$. De manière duale si u n'est pas accepté par $\mathcal{A}$, alors $(\sigma_A \triangleleft \tau_A)^u$ est une stratégie gagnante pour Adam dans le jeu $\mathcal{G}(\mathcal{A} \triangleleft \mathcal{B}, u)$. Il s'ensuit que (a) $\mathcal{A}$ et $\mathcal{A} \triangleleft \mathcal{B}$ acceptent le même langage, et que (b) de plus $\mathcal{A} \triangleleft \mathcal{B}$ est déterministe en histoire. Ainsi, le deuxième point de la proposition est établi.

Composition d'un jeu avec un déterministe en histoire. Considérons maintenant le cas d'un jeu $\mathcal{G}$ (déterminé) et d'un automaton $\mathcal{B}$ déterministe en histoire, et montrons que $\mathcal{G}$ et $\mathcal{G} \triangleleft \mathcal{B}$ sont gagnés par le même joueur. Cela se déduit directement du cas précédent, en effet, un jeu est un automate alternant sur l'alphabet en entrée $\mathbb{I}$. Nous avons vu (remarque 7.22) que le jeu est déterminé si et seulement si l'automate est déterministe en histoire. Nous pouvons donc appliquer le cas précédent, et obtenir que (d'après (b)) $\mathcal{G} \triangleleft \mathcal{B}$ est déterministe en histoire, c'est à dire déterminé, et que (d'après (a)) le langage accepté par $\mathcal{G}$ et $\mathcal{G} \triangleleft \mathcal{B}$ sont les mêmes, *c.-à-d.* que $\mathcal{G}$ et $\mathcal{G} \triangleleft \mathcal{B}$ ont le même vainqueur.

Composition d'un automate alternant avec un déterministe en histoire. Considérons maintenant la composition d'un automate $\mathcal{A}$ avec un automate déterministe en histoire $\mathcal{B}$. Soit

u un mot accepté par $\mathcal{A}$, alors Ève gagne le jeu $\mathcal{G}(\mathcal{A}, u)$. Il s'ensuit que (cas précédent) Ève gagne le jeu $\mathcal{G}(\mathcal{A}, u) \triangleleft \mathcal{B}$. Ce jeu est (à renommage des positions près) égal à $\mathcal{G}(\mathcal{A} \triangleleft \mathcal{B}, u)$ (il s'agit d'une forme d'associativité de la composition syntaxique). Il s'ensuit que Ève gagne le jeu $\mathcal{G}(\mathcal{A} \triangleleft \mathcal{B}, u)$, et par conséquent $\mathcal{A} \triangleleft \mathcal{B}$ accepte le mot u. Dualement, si u n'est pas accepté par $\mathcal{A}$, alors il ne l'est pas non plus par $\mathcal{A} \triangleleft \mathcal{B}$. Ainsi, $\mathcal{A}$ et $\mathcal{A} \triangleleft \mathcal{B}$ acceptent le même langage. CQFD

7.8.2 Composition d'automates à coût

Tout comme pour les automates booléens, il est possible de composer des automates à coût. Nous obtenons la proposition suivante.

Proposition 7.38. *Si $\mathcal{A}$ et $\mathcal{B}$ sont deux automates alternants de coût ayant le même paramètre d'optimisation (*min *ou* max*), et tels que la fonction de coût de $\mathcal{B}$ coïncide avec la fonction de coût de l'objectif d'$\mathcal{A}$, alors :*

- *si $\mathcal{B}$ est déterministe en histoire, alors $\mathcal{A}$ et $\mathcal{A} \triangleleft \mathcal{B}$ acceptent la même fonction de coût,*
- *si $\mathcal{A}$ et $\mathcal{B}$ sont tous deux déterministes en histoire, alors $\mathcal{A} \triangleleft \mathcal{B}$ est déterministe en histoire.*

La preuve est similaire à celle de la Proposition 7.37.

7.8.3 Composition parallèle

Il existe une autre façon de composer deux automates déterministes en histoire, la composition parallèle. Typiquement, dans le cadre des automates déterministes, si nous disposons d'un W-automate déterministe pour le langage L, et d'un W'-automate déterministe pour le langage L', alors il est possible de construire un $W \wedge W'$-automate pour le langage $L \wedge L'$ (*resp.* un $W \wedge W'$-automate pour le langage $L \vee L'$) où

$$W \wedge W' = \{u \times v \ : \ u \in W \text{ et } v \in W'\} \ ,$$
$$\text{et} \quad W \vee W' = \{u \times v \ : \ u \in W \text{ ou } v \in W'\} \ .$$

où $u \times v$ représente le mot sur l'alphabet produit qui, projeté sur la première composante donne u, et projeté sur la deuxième composante donne v.

Une telle construction est possible pour les automates déterministes en histoire. La seule subtilité provient des terminateurs. En effet, le principe est de construire un automate qui exécute de manière concurrente les deux automates dont il est le produit. Si jamais l'exécution de l'un se termine prématurément, il peut être nécessaire de continuer l'exécution de l'autre automate.

Nous commençons par traiter le cas des automates préservant la longueur pour lesquels ce type de complications ne se produit pas.

Lemme 7.39. *Soit $\mathcal{A}$ un W-automate déterministe en histoire pour le langage L préservant la longueur, et L' un autre langage, alors l'automate peut être transformé en un $W \wedge L'$-automate ($W \vee L'$-automate) déterministe en histoire pour le langage $L \wedge L'$ (resp. $L \vee L'$) qui préserve la longueur. De plus, si W est non-déterministe (resp. universel, resp. déterministe), alors l'automate résultat l'est aussi.*

Démonstration. Il suffit d'enrichir l'automate de sorte qu'il copie la deuxième composante de son entrée. Ainsi, il s'agit de prendre l'automate $\mathcal{A}$, de garder ses états et son état initial, et de définir comme fonctions de transitions

$$\delta'(p,(a,b)) = \delta(p,a)[(p,a,c,q) \leftarrow (p,(a,b),(c,b),q)] \ .$$

Il est aisé de vérifier qu'un tel automate a les propriétés annoncées. CQFD

Le résultat est donc que l'on peut simultanément transformer plusieurs conditions.

Lemme 7.40. *Si $\mathcal{A}$ est un W-automate déterministe en histoire préservant la longueur pour le langage L et $\mathcal{A}'$ est un W'-automate déterministe en histoire préservant la longueur pour le langage L', alors il existe un $W \wedge W'$-automate (resp. un $W \vee W'$-automate) $\mathcal{A}''$ déterministe en histoire pour le langage $L \wedge L'$ (resp. $L \vee L'$). De plus, si $\mathcal{A}$ et $\mathcal{A}'$ sont non-déterministes (resp. universel, resp. déterministe), alors l'automate résultat l'est aussi.*

Démonstration. En effet, d'après le lemme précédent, il existe un $W \wedge L'$-automate déterministe en histoire $\mathcal{B}$ pour le langage $L \wedge L'$ et il existe un $W \wedge W'$ automate déterministe en histoire $\mathcal{B}'$ pour le langage $W \wedge L'$. En composant ces deux automates grâce à la proposition 7.37, nous obtenons un $W \wedge W'$-automate déterministe en histoire pour le langage $L \wedge L'$. CQFD

Il nous reste à traiter le cas des automates qui ne préservent pas la longueur.

Proposition 7.41. *Supposons que W et W' sont des conditions ayant la propriété d'élasticité, alors, $\mathcal{A}$ est un W-automate déterministe en histoire pour le langage L et $\mathcal{A}'$ est un W'-automate déterministe en histoire préservant la longueur pour le langage L', alors il existe un $W \wedge W'$-automate $\mathcal{A}''$ déterministe en histoire pour le langage $L \wedge L'$.*

De plus, si $\mathcal{A}$ et $\mathcal{A}'$ sont universels, alors l'automate pour $L \wedge L'$ est aussi universel.

Démonstration. L'idée de la preuve est de se ramener tout d'abord à des automates préservant la longueur, sur lesquels nous pouvons appliquer le résultat précédent. Cela est possible au prix d'enrichir les conditions W et W'. Dans un deuxième temps, une composition avec un automate déterministe en histoire permet de revenir aux conditions W et W' originales. Cet automate déterministe en histoire nécessite l'hypothèse d'élasticité pour être défini.

Ainsi, pour la condition W sur l'alphabet avec terminateur $\mathbb{A}$, il s'agit de construire une nouvelle condition X sur l'alphabet $\mathbb{B}$ dont les caractéristiques sont les suivantes.

1. $\mathbb{B}_0 = \mathbb{A}_1 \cup \mathbb{A}_0^\dagger \cup \{n_1\}$ et $\mathbb{B}_0 = \mathbb{A}_0 \cup \{n_0\}$, où $\mathbb{A}_0^\dagger = \{c^\dagger \ : \ c \in \mathbb{A}_0\}$, et n_0 et n_1 sont des nouveaux symboles.
2. Un mot sur ce nouvel alphabet sera dit **correct** s'il appartient à :
$$\mathbb{A}_1^\omega + \mathbb{A}_1^*\mathbb{A}_0 + \mathbb{A}_1^*\mathbb{A}_0^\dagger n_1^* n_0 + \mathbb{A}_1^*\mathbb{A}_0^\dagger n_1^\omega \ .$$
En d'autres termes, il s'agit des mots sur l'alphabet $\mathbb{A}$, éventuellement prolongés par un mot de $n_1^\omega + n_1^* n_0$, que nous appellerons **complété**. Dans ce cas, le terminateur c est remplacé par sa variante $c^\dagger$. Le mot original est simplement dénoté $\pi(u)$.
3. Un mot correct u est dans X si et seulement si $\pi(u) \in W$.

La condition X' s'obtient de manière similaire.

Il est aisé de transformer un W-automate en un X-automate préservant la longueur. Il suffit pour cela de rallonger les exécutions en se servant des symboles n_1, et n_0. Remarquons que cette construction ne peut que produire des mots corrects.

Soient $\mathcal{B}$ et $\mathcal{B}'$ les automates obtenus de la sorte à partir de $\mathcal{A}$ et $\mathcal{A}'$. Ces automates possèdent toutes les caractéristiques de $\mathcal{A}$ et $\mathcal{A}'$ et en particulier le déterminisme en histoire. Ils acceptents aussi les langages L et L' respectivement.

Nous pouvons donc appliquer le lemme 7.40. Nous obtenons alors un $X \wedge X'$-automate $\mathcal{C}$ pour $L \wedge L'$. Il s'agit de le transformer en un $W \wedge W'$-automate. Pour cela, il nous suffit de produire un $W \wedge W'$-automate pour $X \wedge X'$ déterministe en histoire, et de le composer avec $\mathcal{C}$.

Cet automate est particulièrement simple. Soient $\top_1, \top_0, \top'_1, \top'_0$ les témoins d'élasticité de W et W'. L'alphabet en entrée est $\mathbb{B} \times \mathbb{B}'$. L'automate possède trois états p, g, d. L'état initial est p et signifie qu'il faut suivre à la fois la composante $\mathbb{A}$ et la composante $\mathbb{A}'$ de l'entrée. L'état g signifie qu'il ne reste que la composante $\mathbb{A}$ à suivre, la composante $\mathbb{A}'$ étant déjà entrée dans son complété. L'état d joue le même rôle pour la composante $\mathbb{A}'$. Formellement, la fonction de transition est définie comme suit, pour $a_0 \in \mathbb{A}_0, a_1 \in \mathbb{A}_1, a'_0 \in \mathbb{A}'_0$ et $a'_1 \in \mathbb{A}'_1$ (certains paramètres sont remplacés par $-$ pour simplifie la présentation) :

$$\begin{aligned}
\delta(p, (a_1, a'_1)) &= (-, -, (a_1, a'_1), p) \\
\delta(p, (a_1, {a'_0}^\dagger)) &= (-, -, (a_1, \top'_1), g) \wedge (-, -, (\top_0, a'_0), \bot) \\
\delta(p, ({a_0}^\dagger, a'_1)) &= (-, -, (\top_1, a'_1), d) \wedge (-, -, (a_0, \top'_0), \bot) \\
\delta(p, ({a'_1}^\dagger, {a'_0}^\dagger)) &= (-, -, (a'_1, a'_0), \bot)
\end{aligned}$$

$$\begin{aligned}
\delta(g, (a_1, n_1)) &= (-, -, (a_1, \top'_1), \bot) & \qquad \delta(d, (n_1, a'_1)) &= (-, -, (\top_1, a_1)) \\
\delta(g, ({a_0}^\dagger, n_1)) &= (-, -, (a0, \top'_0), \bot) & \qquad \delta(d, (n_1, {a'_0}^\dagger)) &= (-, -, (\top_0, a'_0), \bot) \\
\delta(g, (a_0, n_0)) &= (-, -, (a0, \top'_0), \bot) & \qquad \delta(d, (n_0, a_0)) &= (-, -, (a0, \top_0), \bot)
\end{aligned}$$

et tous les autres cas sont indéterminés. CQFD

Ces constructions sont tout à fait générales, et nous les utiliserons dans le cadre des jeux de coût comme des jeux de domination (section suivante) sans plus de précaution.

7.9 Jeux de domination finis

Nous avons considéré pour l'instant des jeux de coût, qu'ils soient finis ou infinis. Ils servent à décrire des valeurs entières. D'autre types de jeux sont également à considérer, finis, mais avec des conditions de gain plus complexes : les jeux de domination.

Remarquons que les deux types de jeux (jeux de coût et jeux de domination) sont différents, et utilisés dans des contextes différents. Les deux objets sont nécessaires au développement de la théorie. Les jeux de coût servent à modéliser le comportement d'un automate, et l'étude des stratégies gagnantes sur des arènes quelconques (possiblement infinies) est nécessaire pour prouver la correction de certaines constructions sur les automates. Les jeux de domination quant à eux servent aux procédures de décision, il s'agit de jeux sur des arènes finies, destinés à être représentés en machine, et dont il faut déterminer algorithmiquement le vainqueur. Dans la théorie des mots et des arbres infinis, ces deux types de jeux sont de même nature. Seul le caractère fini de certains diffère. (En fait, à plus haut niveau, quand «l'ordre supérieur» est ajouté aux fonctions de coût, voir piste page 186, ces différents types de jeu sont réunifiés).

Commençons par définir les jeux auxquels nous nous intéressons dans ce cadre. Leurs objectifs sont un peu plus compliqués que dans le jeux de coût. En effet, ces jeux servent à exprimer la domination entre deux fonctions. Ainsi, l'objectif du joueur qui cherche à infirmer cette domination est de trouver un témoin de non domination pour lequel la première fonction n'est pas bornée, alors que la seconde l'est. Le joueur a donc un double objectif : à la fois maximiser une fonction, et en minimiser une autre.

Un **jeu de domination** est un jeu $(V, v_0, \delta, O_{\min}, O_{\max})$ tel que d'une part $(V, v_0, \delta, O_{\min})$ est un jeu de coût dont l'objectif $O_{\min} = (\mathbb{C}, f, \min)$ est à minimiser, et d'autre part $(V, v_0, \delta, O_{\max})$ est un jeu de coût dont l'objectif $O_{\max} = (\mathbb{C}, g, \max)$ est à maximiser. Nous dirons que Ève **gagne le jeu de domination** s'il existe un entier n tel que pour tout entier m il existe une stratégie pour Ève σ_{E} qui gagne pour $(O_{\min})_n \cap (O_{\max})_m$. Nous ferons aussi référence à un tel jeu comme à un jeu d'**objectif de domination** $O_{\min} \wedge O_{\max}$.

Ce que nous allons voir dans ce chapitre, c'est la décidabilité du cas fini.

Proposition 7.42 (extension de [33])**.** *Le vainqueur des jeux de domination finis ayant des fonctions régulières de coût comme objectifs est décidable.*

En utilisant le théorème 7.35 nous savons que la fonction à minimiser se décrit au moyen d'un $\mathtt{B} \wedge \mathtt{parité}$-automate déterministe en histoire, et la fonction à maximiser au moyen d'un $\mathtt{S} \wedge \mathtt{parité}$-automate déterministe en histoire. En composant ces deux automates avec le jeu, il en résulte un jeu de domination $(\mathtt{B} \wedge \mathtt{parité}) \wedge (\mathtt{S} \wedge \mathtt{parité})$. En le combinant avec le corollaire 7.33, ce jeu devient un jeu de domination $(\mathtt{B} \wedge \mathtt{parité}) \wedge (\neg\mathtt{B} \wedge \mathtt{parité})$. La

conjonction de conditions de parité étant une condition de `Streett`, le jeu résultant peut être qualifié de $\mathtt{B} \wedge \neg\mathtt{B} \wedge \mathtt{Streett}$. Ainsi, pour gagner un tel jeu, Ève doit donner un entier m tel que pour tout n, il existe une stratégie σ_{E} gagnante pour la condition $\mathtt{B}_m \wedge \neg\mathtt{B}_n \wedge \mathtt{Streett}$, c'est à dire que toute partie satisfait :

B : aucun des B-compteurs[2] ne dépasse la valeur m,

¬B : si jamais un terminateur 0 pour la condition ¬B (nous le noterons $0_{\neg\mathtt{B}}$dans la suite[3]) est rencontré, alors un ¬B-compteur a dépassé la valeur n auparavant (il a pu être remis à zéro depuis),

Streett : la condition `Streett` est satisfaite.

Soit $\mathcal{G}$ un tel jeu fini, qu'il s'agit de résoudre. Nous allons le transformer en un jeu de Muller $\mathcal{G}'$ (que nous prouverons de même vainqueur). Ce jeu $\mathcal{G}'$ s'obtient à partir de $\mathcal{G}$ de la manière suivante :

- Toutes les positions sont dupliquées, ce qui revient à les enrichir d'un bit, appelé dans la suite «**le mode**». Intuitivement, ce mode peut être pensé à 1 si l'un des compteurs de la condition ¬B à un moment antérieur de la condition, a dépassé la valeur n.
- La composante correspondant au jeu original évolue comme dans le jeu original, les mêmes actions sont générées (en revanche, leur sens change puisque la nouvelle condition d'acceptation est de Muller), et les joueurs ont le même contrôle sur le déroulement de la partie.
- Le mode reste inchangé durant la partie, si ce n'est que, à tout moment, Adam a la possibilité de faire passer le mode à 1. Ainsi, soit le mode reste éternellement 0, soit il passe à 1 et y reste jusqu'à la fin de la partie.
- L'état initial est celui du jeu original dans le mode 0.
- Si l'action $0_{\neg\mathtt{B}}$ est rencontrée alors que le mode est 0, Ève perd ; si le mode est a 1, Ève gagne.
- La condition de gain sur les parties infinies est de la forme $\mathtt{B}' \wedge (\neg\mathtt{B}' \vee \mathtt{Streett})$ où :

  B′ : Pour tout B-compteur, s'il est incrémenté une infinité de fois, alors il est aussi remis à zéro une infinité de fois.

  ¬B′ : Il existe un ¬B-compteur qui est incrémenté une infinité de fois, remis à zéro un nombre fini de fois, et le compteur reste à zéro.

  Streett : Il s'agit de la condition `Streett` du jeu original.

Il s'agit d'un jeu de Muller. Il peut donc être résolu par des techniques standards. De plus, nous utiliserons le résultat classique d'existence de stratégie à mémoire finie. Nous montrons dans la suite que ce jeu a le même vainqueur que le jeu de domination original $\mathcal{G}$.

2. Nous allégeons les notations et parlons informellement de B-compteurs et de ¬B-compteurs. Il s'agit bien entendu des compteurs impliqués dans l'objectif correspondant.

3. D'autres terminateur peuvent être rencontrés, mais leur rôle n'est pas essentiel et nous n'y faisons pas référence au cours de la preuve. En particulier, ils peuvent être remplacés par des boucles. Par exemple, $\infty_{\neg\mathtt{B}}$ peut être remplacé par une boucle d'actions ε sur tous les compteurs de la condition ¬B. Ce n'est pas le cas pour $0_{\neg\mathtt{B}}$.

Lemme 7.43. *Le jeu $\mathcal{G}'$ a le même vainqueur que le jeu $\mathcal{G}$.*

Démonstration. Première direction. Supposons qu'Ève possède une stratégie gagnante dans le jeu $\mathcal{G}'$, et montrons qu'Ève gagnait aussi le jeu original $\mathcal{G}$. Remarquons tout d'abord que le jeu $\mathcal{G}'$ est un jeu de Muller, et par conséquent, Ève le gagne au moyen d'une stratégie à mémoire finie σ'_{E}. Soit m le produit de la taille du jeu $\mathcal{G}'$ et de la mémoire de la stratégie σ'_{E} (*c.-à-d.* le nombre de sous-arbres non-isomorphes de la stratégie). Fixons-nous un entier n. Il s'agit de produire une stratégie σ^n_{E} gagnante dans le jeu $\mathcal{G}$ pour la condition $\mathtt{B}_m \wedge \neg\mathtt{B}_n \wedge \mathtt{Streett}$.

Cette stratégie s'obtient en transformant σ'_{E}. Pour cela, nous fixons les choix d'Adam dans le jeu $\mathcal{G}'$ qui n'ont pas d'équivalent dans le jeu $\mathcal{G}$, *c.-à-d*, nous précisons quand Adam décide de changer le mode. Cela se fait comme suit :

(*) Adam décide de faire passer le mode à 1 aussitôt que l'un des compteurs de la condition $\neg\mathtt{B}$ dépasse la valeur n.

En fixant ce choix dans la stratégie σ'_{E}, puis en projetant sur le jeu $\mathcal{G}$, nous obtenons notre stratégie σ^n_{E}.

Montrons que σ^n_{E} est gagnante pour le condition $\mathtt{B}_m \wedge \neg\mathtt{B}_n \wedge \mathtt{Streett}$. Considérons donc une partie π compatible avec σ^n_{E}, et soit π' la partie compatible avec σ'_{E} dont elle est la projection.

$\mathtt{B}_m$ Supposons que π ne satisfasse pas la condition $\mathtt{B}_m$, cela signifie qu'il existe une portion d'exécution qui contient plus de m incréments d'un des compteurs de la condition $\mathtt{B}$, et aucune remise à zéro de ce compteur. Nécessairement, par choix de m, cela signifie qu'il existe un cycle dans la stratégie σ'_{E} au cours duquel l'un des compteurs de l'objectif $\mathtt{B}$ est incrémenté sans jamais être remis à zéro. En répétant infiniment ce cycle, nous produisons une partie compatible avec σ'_{E} au cours de laquelle ce compteur est incrémenté une infinité de fois, et n'est remis à zéro qu'un nombre fini de fois. L'existence de cette branche contredit le fait que σ'_{E} est gagnante, en particulier pour la condition $\mathtt{B}'$.

$\neg\mathtt{B}_n$ Supposons que π ne satisfasse pas la condition $\neg\mathtt{B}_n$. Cela signifie que π s'achève par le terminateur $0_{\neg\mathtt{B}}$, mais qu'aucun $\neg\mathtt{B}$-compteur ne dépasse la valeur n. D'après la définition de σ^n_{E} (*), cela signifie que le mode ne passe jamais à 1 au cours de cette exécution. Ainsi, le mode est à 0 quand le terminateur $0_{\neg\mathtt{B}}$ est rencontré. Cela contredit le fait que σ'_{E} est gagnante dans $\mathcal{G}'$.

$\mathtt{Streett}$ Montrons tout d'abord que π' ne satisfait pas $\neg\mathtt{B}'$. En effet, s'il existe un $\neg\mathtt{B}$-compteur qui est incrémenté une infinité de fois et n'est remis à zéro qu'un nombre fini de fois, alors le compteur inéluctablement dépasse la valeur n, et par conséquent le mode passe à 1 (*). Ainsi, π' ne peut satisfaire $\neg\mathtt{B}'$. Or, comme σ'_{E} est gagnante, π' satisfait $\neg\mathtt{B}' \vee \mathtt{Streett}$, et par conséquent $\mathtt{Streett}$. Comme π est la projection de π', π satisfait aussi $\mathtt{Streett}$.

Seconde direction. Supposons qu'Adam gagne le jeu $\mathcal{G}'$, et soit σ'_{A} une stratégie gagnante à

mémoire finie. Notre objectif est de montrer que pour tout entier m, il existe un entier n et une stratégie pour Adam σ_A dans le jeu $\mathcal{G}$ gagnante pour (en fait, contre) la condition $\mathtt{B}_m \wedge \neg\mathtt{B}_n \wedge \mathtt{Streett}$. Ainsi, nous supposons que m est fixé dans la suite. Soit $n = mk$, où k est le nombre de sous-arbres distincts de la stratégie σ'_A. La stratégie pour Adam σ_A est simplement la projection de σ'_A sur le jeu $\mathcal{G}$ (remarquons qu'elle ne dépend pas de m).

Il nous reste à montrer que cette stratégie est gagnante dans le jeu $\mathcal{G}$ pour (en fait contre) la condition $\mathtt{B}_m \wedge \neg\mathtt{B}_n \wedge \mathtt{Streett}$. Pour cela, nous raisonnons par la contraposée. Nous supposons qu'une partie π compatible avec σ_A soit gagnante pour Ève. Soit de plus π' la partie compatible avec σ'_A dont π est la projection. Notre objectif est de contredire l'hypothèse que la stratégie σ'_A est gagnante pour Adam. Deux cas essentiellement sont à considérer.

Si π s'achève par le terminateur $0_{\neg\mathtt{B}}$.

– Si le mode est à 1 à ce moment, π' est gagnante pour Ève dans $\mathcal{G}'$ par définition du jeu,

– Sinon, le mode est à 0 au moment de la rencontre du terminateur $0_{\neg\mathtt{B}}$. Comme la partie π' est supposée gagnante pour $\neg\mathtt{B}$, cela signifie que l'un des $\neg\mathtt{B}$-compteurs a dépassé la valeur n. Comme n a été choisi suffisamment grand devant m, k, et σ'_A, il existe un facteur non-trivial de π' qui : (a) forme un cycle de la stratégie σ'_A, *c.-à-d*, commence et se termine dans la même position de $\mathcal{G}'$ et le même état mémoire, (b) au cours du cycle, tout $\mathtt{B}$-compteur est soit remis à zéro, soit n'est jamais incrémenté, et (c) l'un des $\neg\mathtt{B}$-compteurs est incrémenté au moins une fois, et n'est jamais remis à zéro. Soit donc π'' la partie obtenue de π' en itérant infiniment souvent le cycle décrit ci-dessus. Cette partie est compatible avec σ'_A par (a). De plus, π'' satisfait

$\mathtt{B}'$ par itération de (b), et

$\neg\mathtt{B}'$ En effet, par itération de (c), il existe un $\neg\mathtt{B}$-compteur qui est incrémenté infiniment souvent et n'est remis à zéro qu'un nombre fini de fois dans π''. Comme de plus le mode reste à 0 au cours de cette exécution, la condition $\neg\mathtt{B}'$ est satisfaite.

Ainsi, π'' est gagnante pour Ève, contredisant le fait que σ'_A est gagnante pour Adam.

Sinon, si π ne s'achève pas par le terminateur $0_{\neg\mathtt{B}}$, et π' satisfait

$\mathtt{B}'$ car π satisfait $\mathtt{B}_m$, et que $\mathtt{B}_m$ implique $\mathtt{B}'$, et

$\mathtt{Streett}$ car π satisfait déjà $\mathtt{Streett}$.

Ainsi, π' contredit le fait que σ'_A est gagnante.

CQFD

Corollaire 7.44. *Déterminer le vainqueur d'un jeu de domination ayant des fonctions régulières de coût comme objectif est un problème décidable.*

Démonstration. Décider du vainqueur d'un tel jeu se réduit à décider du vainqueur dans un jeu de Muller fini, d'après le lemme 7.43. Ce dernier problème est décidable. CQFD

Ce résultat de décidabilité conclut ce chapitre. Le chapitre suivant met en action toutes ces techniques de jeu dans des procédures de décidabilité de la logique monadique de coût sur les arbres.

Chapitre 8

Les arbres

Dans ce chapitre, nous montrons comment décider la logique monadique de coût (la propriété de domination) sur les arbres finis. Ce chapitre est l'occasion de mettre en action les résultats du chapitre précédent (chapitre 7). Il s'agit de collaborations avec Christof Löding [31, 33]. Beaucoup d'objets sont introduits dans le cadre plus général des arbres infinis, bien que le résultat complet de décidabilité ne soit valable que pour les arbres finis. Nous discuterons de ce point au cours du chapitre suivant qui traite des arbres infinis.

Ce chapitre commence par une motivation, le problème de la hauteur d'étoile sur les arbres. Il s'agit de la section 8.1. Les automates d'arbres (même infinis) sont introduits à la section 8.2. Au cours de la section 8.3 nous montrons comment décider de la domination entre fonctions régulières de coût sur les arbres finis quand elles sont représentées par des automates de la bonne forme. Le résultat central de simulation, permettant de transformer les automates d'arbres finis alternants en automates non-déterministes, est établi au cours de la section 8.4. Il permet en particulier d'établir le théorème fondamental des fonctions régulières de coût sur les arbres finis (théorèmes 8.12 et 8.13).

8.1 La hauteur d'étoile sur les arbres

Nous avons vu que l'une des motivations à l'introduction des B-automates hiérarchiques était de décider de problèmes comme la hauteur d'étoile [52]. Une question similaire (en fait plusieurs) se posent naturellement sur les arbres. Le bon cadre pour formaliser ces questions est, plutôt que les expressions régulières, le μ-calcul (nous utilisons ici une variante du μ-calcul qui ne peut utiliser le plus grand point fixe : en effet, le plus grand point-fixe n'apporte rien quand il s'agit de définir des langages d'arbres finis). Pour cette raison, quelques définitions sont nécessaires (tout comme le reste de cette section, elles ne seront plus rencontrées ailleurs dans ce document). Nous supposons pour la suite un alphabet gradué $\mathbb{A}$ fixé.

Une formule du **μ-calcul disjonctif avec substitution** (plus simplement une μ-

formule avec substitution) a la syntaxe suivante :

$$\phi ::= \quad \bot \quad | \quad a(\underbrace{\phi, \ldots, \phi}_{k \text{ sous-formules}}) \quad | \quad \phi + \phi \quad | \quad x \quad | \quad \mu x.\phi \quad | \quad \phi[x := \phi] \ ,$$

dans laquelle $a \in \mathbb{A}$ est une lettre d'arité k, et x est une **variable**. Si a est une constante (d'arité 0), elle sera juste notée a au lieu de $a()$. Si une μ-formule avec substitution n'utilise pas la construction $\phi[x := \phi]$, elle est simplement appelée une μ**-formule**. Une μ-formule est dite **fermée** si elle ne contient pas de variables libres.

La sémantique $[\![\phi]\!]$ d'une μ-formule est le langage d'arbres défini par :

- $[\![\bot]\!] = \emptyset$,
- $[\![x]\!] = \{x\}$,
- $[\![a(\psi_1, \ldots, \psi_k)]\!] = \{a(t_1, \ldots, t_k) \ : \ t_1 \in [\![\phi_1]\!], \ldots, t_k \in [\![\phi_k]\!]\}$, (où $a([\![\psi_1]\!], \ldots, [\![\psi_k]\!])$ pour simplifier) pour a d'arité k,
- $[\![\psi + \psi']\!] = [\![\psi]\!] \cup [\![\psi']\!]$,
- $[\![\phi[x := \psi]]\!] = [\![\phi]\!][x := [\![\psi]\!]]$, où $L[x := K]$ représente l'ensemble des arbres obtenus d'un arbre $t \in L$ en substituant à chaque occurrence de x dans t un arbre de K,
- $[\![\mu x.\psi]\!] = \bigcup_{n \in \mathbb{N}} L_n$, où $L_0 = \emptyset$ et $L_{n+1} = L_n \cup [\![\psi]\!][x := L_n]$.

Ainsi, l'ensemble des arbres sur l'alphabet $\{f : 2, a : 0, b : 0\}$ est décrit par la μ-formule :

$$(\mu x.f(x, x) + a + b) \ .$$

L'ensemble des arbres sur cet alphabet qui contiennent un nombre pair d'occurrences de la lettre b s'exprime par la μ-formule suivante :

$$\begin{aligned} \mu x. \ & f((\mu y. \ f(x, y) + f(y, x) + b), (\mu y. \ f(x, y) + f(y, x) + b)) \\ & + f(x, x) + a \ . \end{aligned}$$

Une façon de comprendre cette formule est de voir la variable x comme les arbres contenant un nombre pair de b, et y comme ceux contenant un nombre impair de b. Notons X l'ensemble des arbres contenant un nombre pair d'occurrences de b, et Y ceux qui en contiennent un nombre impair. Nous avons bien que $X = f(Y, Y) \cup f(X, X) \cup \{a\}$, et même plus : X est le plus petit ensemble qui satisfait cette équation. Ainsi, X se décrit comme $\mu x.f(Y, Y) + f(x, x) + a$. De même Y est le plus petit ensemble tel que $Y = f(X, Y) \cup f(Y, X) \cup b$. Ainsi, Y se décrit comme $\mu y.f(X, y) + f(y, X) + b$. La formule combine ces deux points fixes en substituant à chaque occurrence de Y dans $\mu x.f(Y, Y) + f(x, x) + a$ sa définition $\mu y.f(x, y) + f(y, x) + b$.

Il est classique qu'une μ-formule avec substitution définit un langage régulier d'arbre, et réciproquement tout langage régulier d'arbre fini peut-être décrit au moyen d'une d'une μ-formule (sans substitution).

Le **niveau d'imbrication** $\texttt{ni}(\phi)$ d'une μ-formule ϕ (avec ou sans substitution) est le plus grand nombre de points fixes imbriqués. Ainsi,

- $\mathtt{ni}(\bot) = \mathtt{ni}(x) = 0$,
- $\mathtt{ni}(\phi + \phi') = \mathtt{ni}(\phi[x := \psi]) = \max(\mathtt{ni}(\phi), \mathtt{ni}(\psi))$,
- $\mathtt{ni}(a(\phi_1, \ldots, \phi_k)) = \max(\mathtt{ni}(\phi_1), \ldots, \mathtt{ni}(\phi_k))$, pour a d'arité k, et
- $\mathtt{ni}(\mu x.\phi) = \mathtt{ni}(\phi) + 1$.

Remarque 8.1. En fait, une définition de la notion d'expression régulière existe. Pour plus de précisions, nous orientons le lecteur vers le deuxième chapitre du livre [34]. Cette définition induit une hauteur d'étoile comme pour les langages de mots. Ce paramètre est relié au niveau d'imbrication comme suit : tout langage régulier peut être défini par une μ-formule avec substitution de niveau d'imbrication k si et seulement si elle peut être définie au moyen d'une expression régulière de hauteur d'étoile k. Ainsi, résoudre le problème de la hauteur d'étoile sur les arbres est équivalent à déterminer le niveau d'imbrication d'un langage parmi les formules avec substitution. La question similaire sans les substitutions ne s'exprime pas en terme d'expressions régulières.

En utilisant des techniques inspirées de Kirsten [52], elles même inspirées des travaux de Hashiguchi, nous obtenons le résultat de réduction suivant.

Lemme 8.2 ([31])**.** *Étant donné un langage d'arbres finis L et un entier k,*

1. *il existe effectivement un* B*-automate d'arbre fini qui est borné si et seulement si L est accepté par une μ-formule avec substitution de niveau d'imbrication k, et*
2. *il existe effectivement un* B*-automate d'arbre fini qui est borné si et seulement si L est accepté par une μ-formule de niveau d'imbrication k.*

Nous allons voir dans la suite de ce chapitre les définitions précises des automates à coût sur les arbres ainsi que les techniques pour les résoudre, et en particulier le résultat de décidabilité de la domination. Le théorème 8.12 nous permet, entre autre, de conclure :

Théorème 8.3 ([31])**.** *Les problèmes, étant donné un langage régulier d'arbres finis L et un entier k,*

1. *de déterminer si L est accepté par une μ-formule avec substitution de niveau d'imbrication k, et*
2. *de déterminer si L est accepté par une μ-formule sans substitution de niveau d'imbrication k,*

sont tous deux décidables.

8.2 Automates d'arbres

Nous définissons les automates d'arbre dans toute leur généralité, c'est à dire pour tous les objectifs, dans le cas fini ou infini, et alternants. Nous nous attardons dans cette section aux propriétés que ces automates ont indépendamment de la condition d'acceptation qu'ils utilisent. Remarquons que ces automates généralisent ceux déjà rencontrés sur les mots à

la section 7.6, un alphabet avec terminateur étant un cas particulier d'alphabet gradué où seules les deux arités 0 et 1 sont possibles.

Fixons un alphabet gradué $\mathbb{A}$. Un **automate d'arbre (alternant)** de coût $\mathcal{A}$ est décrit par la donnée de

- un **ensemble fini d'états** Q,
- un alphabet gradué $\mathbb{A}$,
- un **état initial** $q_0 \in Q$,
- un **objectif** $O = (\mathbb{C}, f, opt)$,
- une **fonction de transition** δ qui à tout état q et à toute lettre $a \in \mathbb{A}$ d'arité k associe une combinaison booléenne positive

$$\delta(q,a) \in \mathcal{B}^+(\{1,\dots,k\} \times \mathbb{C}_1 \times Q \ \cup \ \{\bot\} \times \mathbb{C}_0 \times \{\bot\}) \ .$$

(Là encore, l'usage de $\bot$ est une commodité de notation permettant d'unifier les cas.) Chacun des triplets (i,c,q) impliqués dans la combinaison booléenne s'interprète comme «effectuer l'action c, et continuer vers le $i$ème fils du nœud courant, avec l'état q.» Ainsi, dans le cas particulier de la fin d'une exécution, $i = \bot$ signifie que l'exécution ne continue dans aucun fils, et $q = \bot$ signifie qu'aucun état ne permet de continuer le calcul.

Un automate d'arbre est **non-déterministe** s'il est décrit par un ensemble de **transitions**

$$\Delta \subseteq \{(p,a,c_1,q_1,\dots,c_k,p_k) \ : \ a \in \mathbb{A}_k, \ k \geq 1, \ p,q_1,\dots q_k \in Q, \ c_1,\dots c_k \in \mathbb{C}_1\}$$

et une **application finale** F de $Q \times \mathbb{A}_0$ dans $\mathbb{C}_0$. L'automate alternant correspondant est décrit au moyen de la fonction de transition suivante pour a d'arité $k \geq 1$,

$$\delta(p,a) = \bigvee_{(p,a,c_1,q_1,\dots,c_k,p_k)\in\Delta} (1,c_1,q_1) \wedge \dots \wedge (k,c_k,q_k) \ ,$$

et $\delta(p,a) = (\bot, F(p,a), \bot)$ si a est une constante.

Le **dual** d'un automate d'arbre s'obtient en dualisant l'objectif ainsi que les formules booléennes impliquées dans les transitions.

Comme pour les automates alternants sur les mots, la sémantique d'un automate appliqué à un arbre t (sur l'alphabet approprié) est décrite en produisant un jeu. Ainsi, nous définissons le jeu $\mathcal{G}(\mathcal{A},t)$ qui a les caractéristiques suivantes :

- l'ensemble de positions est $V = dom(t) \times Q$, *c.-à-d.* l'ensemble des paires (nœud,état),
- l'objectif est O,
- la position initiale est (ε, q_0) (rappelons que ε représente la racine de l'arbre),
- l'ensemble de coups est défini par

$$M_1 = \{(x,p),c,(y,q) \in V \times \mathbb{C}_1 \times V \ : \ x \text{ parent de } y\}$$
$$M_0 = V \times \mathbb{C}_0 \times \{\bot\} \ ,$$

– l'application de contrôle est définie par

$$\delta(x,p) = \delta_{t(x)}(p)[(c,q,i) \leftarrow ((x,p),c,(xi,q))] \ .$$

Comme dans le cas des mots, la valeur calculée par l'automate sur l'arbre t en entrée est :

$$[\![\mathcal{A}]\!](t) = \mathit{valeur}(\mathcal{G}(\mathcal{A},t)) \ .$$

Comme nous aurons le plus souvent besoin de manipuler des automates non-déterministes, il est pratique de disposer de notations adaptées pour en décrire le comportement. Ainsi, il est plus pratique de parler d'**exécutions** que de stratégies. L'équivalence entre les deux notions est standard. Considérons donc un automate d'arbre non-déterministe décrit par un ensemble d'état Q, un état initial q_0, un ensemble de transitions Δ, une application finale F, et un objectif O. Une **exécution** est une application σ des nœuds (non-feuilles) de l'arbre dans les transitions de l'automates qui respecte certaines contraintes locales. En pratique nous utiliserons souvent une paire d'applications σ, ρ, où ρ peut être reconstruite à partie de σ. L'application σ associe à chaque nœud interne de l'arbre une transition, et ρ associe à chaque nœud (non-nécessairement interne) un état. Elles doivent respecter les contraintes suivantes :

– pour tout nœud interne x, si $\sigma(x) = (q, a, c_1, q_1, \ldots, c_k, q_k)$, alors $t(x) = a$, $\rho(x) = q$, et $\rho(xi) = q_i$ pour tout $i = 1 \ldots k$,
– $\rho(\varepsilon) = q_0$.

Il s'agit maintenant d'associer une valeur à une exécution. Étant donnée une exécution (σ, ρ) et une branche $B = k_1 k_2 k_3 \ldots$ (finie ou infinie), construisons le mot avec terminateur $\sigma[B] = d_1 d_2 \ldots$ sur $\mathbb{C}$ qui dénote la suite des lettres de $\mathbb{C}$ rencontrées par l'exécution le long de la branche B, et se concluant éventuellement par un terminateur obtenu par F. Formellement, soit $x_i = k_1 k_2 \ldots k_i$ (c'est à dire le $i$ème nœud de la branche, la racine étant $x_0 = \varepsilon$), posons pour tout $i \geq 1$,

– si x_i est une feuille alors $d_i = F(\rho(x_i), t(x_i))$,
– sinon $d_i = c_{k_i}$, où $\sigma(x_i) = (p, a, c_1, q_1, \ldots, c_k, q_k)$.

Si $\mathit{opt} = \min$ (*resp.* $\mathit{opt} = \max$), alors la valeur de l'exécution $\mathit{valeur}(\sigma)$ est le supremum (*resp.* l'infimum) sur toutes les branches B de $f(\sigma[B])$. La valeur calculée par l'automate sur l'arbre t est alors :

$$[\![\mathcal{A}]\!](t) = \begin{cases} \sup\{\mathit{valeur}(\sigma) \ : \ \sigma \text{ exécution de } \mathcal{A} \text{ sur } t\} & \text{si } \mathit{opt} = \min \\ \inf\{\mathit{valeur}(\sigma) \ : \ \sigma \text{ exécution de } \mathcal{A} \text{ sur } t\} & \text{sinon.} \end{cases}$$

La valeur calculée ainsi coïncide avec celle définie en passant par les jeux. En fait, il y a une bijection naturelle entre les stratégies pour Ève sur le jeu induit et les exécutions de l'automate sur un arbre.

Exemple 8.4. Nous présentons dans cet exemple un automate d'arbre non-déterministe utilisant l'objectif `B` à 1 compteur, sur les arbres éventuellement infinis, qui, à $\approx$-près accepte le nombre d'occurrence d'une certaine lettre.

Pour ne pas rendre l'exemple plus compliqué que nécessaire, nous ne cherchons à «compter» que le nombre d'occurrences d'une *feuille* a. Il est élémentaire de généraliser cette construction au fait de compter le nombre d'occurrences d'un nombre quelconque de lettres, possiblement d'arité supérieure à 1. Ainsi, considérons donc l'alphabet gradué $\mathbb{A}$ contenant deux constantes a, b, ainsi qu'un symbole binaire f. Notre objectif est de compter le nombre d'occurrences de la lettre a.

Notre automate possède deux états p et q, ce dernier étant initial, et un unique compteur qui est soit laissé inchangé (ε), soit incrémenté et testé (`ic`) (il s'agit donc d'un automate alternant de distance). L'ensemble de transitions est le suivant :

$$\begin{aligned}\Delta = \ \{ \quad & (p, f, \varepsilon, p, \varepsilon, p),\\ & (q, f, \varepsilon, q, \varepsilon, p),\\ & (q, f, \varepsilon, p, \varepsilon, q),\\ & (q, f, \texttt{ic}, q, \texttt{ic}, q) \ \}\end{aligned}$$

et l'application

$$\begin{aligned} F(p,a) &= \infty \ , & F(q,a) &= 0 \ ,\\ F(p,b) &= 0 \ , & F(q,b) &= 0 \ . \end{aligned}$$

Remarquons tout d'abord que pour toute exécution (σ, ρ), si $\rho(x) = p$ et $t(x) = a$ pour une feuille x, alors la valeur de la branche, et donc de l'exécution, est ∞. Il s'ensuit que seules les exécutions pour lesquelles cette situation ne se produit pas sont intéressantes (l'objectif étant de minimiser la valeur de l'exécution). Remarquons également qu'une exécution associant l'état q à un nœud x, associe l'état q également à tous les ancêtres de x.

Considérons maintenant une exécution de l'automate sur un arbre t. Supposons que toute les branches ont une valeur au plus n. Cela signifie que la transition $(q, f, \texttt{ic}, q, \texttt{ic}, q)$ (la seule qui incrémente le compteur) est utilisée au plus n fois sur chaque branche. Comme cette transition est la seule qui augmente le nombre de branches portant la lettre q, il s'ensuit que l'exécution possède au plus 2^n branches portant uniquement l'état q. Comme toutes les occurrences de la constante a correspondent à des feuilles auxquelles l'exécution associe l'état q, nous en déduisons que l'arbre possède au plus 2^n occurrences de la constante a.

Réciproquement, considérons un arbre possédant n occurrences de la constante a, et construisons l'exécution qui à chaque nœud x associe l'état q s'il existe une occurrence de la lettre a portée par un descendant de x, et l'état p sinon. Nous pouvons vérifier qu'une et une seule transition de l'automate à chaque nœud permet de compléter cette exécution. Vérifions aussi qu'au plus $n-1$-occurrences de la lettre $(q, f, \texttt{ic}, q, f, \texttt{ic}, f)$ ne peuvent apparaître sur une branche de cette exécution. Ainsi la valeur de cette exécution est au plus $n-1$.

Il s'ensuit que cet automate compte le nombre d'occurrences de la lettre a, modulo $\approx$.

Remarquons que la fonction de correction permettant de relier la fonction calculée par l'automate au nombre d'occurrences de la lettre a est exponentielle. Il s'agit de l'unique cas d'une fonction de correction non-polynomiale que nous aurons l'occasion de rencontrer dans ce document.

Avant de rentrer dans le vif du sujet, nous présentons dans cette section quelques résultats élémentaires génériques sur les automates d'arbre (possiblement sur les arbres infinis). Beaucoup sont des conséquences directes des définitions, comme le lemme suivant pour les automates alternants.

Lemme 8.5. *Les fonctions calculées par automates alternants sont closes sous* min *et* max.

ou encore cet autre pour les automates non-déterministes.

Lemme 8.6. *Les classes des fonctions calculées par automates non-déterministes ayant un objectif à minimiser sont closes sous* min *et* inf*-projection. Si la classe d'objectifs est de plus close sous* max*, alors cette classe est aussi close sous* max.

Les fonctions calculées par automates non-déterministes ayant un objectif à maximiser sont closes sous max *et* sup*-projection. Si la classe d'objectifs est de plus close sous* min*, alors cette classe est aussi close sous* min.

L'idée de ces preuves est absolument naturelle : il s'agit de mimer les constructions classiques. Remarquons par exemple, en suivant cet énoncé, que les fonctions calculées par les automates de distance sont closes sous min. En revanche, elles ne le sont pas, en général, sous max. À titre d'exemple, la famille des objectifs de distance n'est pas close sous max (en fait, elles le sont à $\approx$ près), et en effet, les automates de distance ne sont pas clos sous max. Les `B`-automates, en revanche, sont clos sous max car les conditions `B` sont closes sous max (il suffit de prendre l'union disjointe des compteurs). Il n'en va pas de même pour les automates d'objectif `hB` qui ne sont clos sous max que à $\approx$-près. Nous ne développons pas plus avant ces preuves qui sont issues de techniques complètement standards.

Il est un peu plus intéressant d'utiliser les propriétés de clôtures héritées de la notion de déterminisme en histoire. En effet, le déterminisme en histoire permet de composer un jeu avec un automate de mot. Comme la sémantique d'un automate d'arbre passe par l'utilisation d'un jeu, elle se prête à ce type de constructions. Une analyse plus fine révèle que la construction de composition peut être réalisée directement sur l'automate d'arbre. Cela nous donne, par exemple, les résultats suivants.

Lemme 8.7. *Étant donné un automate alternant d'arbre $\mathcal{A}$ dont l'objectif est $(\mathbb{B}, f, \min)$ (resp.* max*), et un automate alternant de mots déterministe en histoire $\mathcal{B}$ qui reconnaît la fonction de coût f et d'objectif $(\mathbb{C}, g, \min)$ (resp.* max*), alors il existe effectivement un automate alternant d'arbre $\mathcal{C}$ équivalent à $\mathcal{A}$ d'objectif $(\mathbb{C}, g, \min)$ (resp.* max*).*

De plus, si $\mathcal{A}$ et $\mathcal{B}$ sont non-déterministes et $\mathcal{B}$ préserve la longueur, alors $\mathcal{C}$ est non-déterministe.

Ainsi, en combinant le lemme ci-dessus avec la proposition 7.29, tout automate alternant d'arbre utilisant une condition d'acceptation `S` peut être transformé en un automate alternant utilisant la condition d'acceptation $\neg$`B`. Soulignons le fait que, même si l'automate de départ est non-déterministe, en général, l'automate d'arrivée ne l'est pas. Cela vient du fait que l'automate de la proposition 7.29 est universel. Également, en utilisant ce résultat en combinaison avec la proposition 7.30, tout automate utilisant une condition d'acceptation `B` (*resp.* $\neg$`B`) est équivalent à un automate utilisant une condition d'acceptation `hB` (*resp.* $\neg$`hB`). De plus, si cet automate est non-déterministe, alors l'automate résultat est aussi non-déterministe. En utilisant ce type d'arguments, tous les automates de ce travail peuvent être mis sous la forme de `hB` $\wedge$ `parité`-automates alternants, ou de $\neg$`hB` $\wedge$ `parité`-automates alternants.

Une autre application des automates déterministes en histoire sur les mots est pour la construction d'automates d'arbre qui vérifient une même propriété sur toutes les branches de l'arbre. Étant donné un arbre t sur l'alphabet gradué $\mathbb{A}$, un **mot de branche** de l'arbre est un mot de la forme

$$a_0 k_1 a_1 k_2 \dots$$

où $k_1 k_2 \cdots \in \mathbb{N}^* \cup \mathbb{N}^\omega$ est une branche de l'arbre, et $a_i = t(k_1 k_2 \dots k_i)$ pour tout i, *c.-à-d.* que a_i est l'étiquette de l'arbre à la position indiquée par le chemin $k_1 \dots k_i$. Ce mot est soit infini, soit fini, et dans ce cas, sa dernière lettre appartient à $\mathbb{A}_0$.

Lemme 8.8. *Étant donné un automate de mot alternant déterministe en histoire $\mathcal{A}$ avec objectif à minimiser (resp. à maximiser) et calculant la fonction f, il peut être transformé en un automate d'arbre $\mathcal{A}^{\sup}$ (resp. $\mathcal{A}^{\inf}$), ayant le même nombre d'états et le même objectif et qui calcule la fonction de coût $f^{\sup}$ (resp. $f^{\inf}$) sur les arbres définie par*

$$f^{\sup}(t) = \sup\{f(u) \ : \ u \text{ branche de } t\}$$
$$\text{(resp.} \quad f^{\inf}(t) = \inf\{f(u) \ : \ u \text{ branche de } t\} \text{).}$$

Si l'automate $\mathcal{A}$ est non-déterministe, alors $\mathcal{A}^{\sup}$ (resp. $\mathcal{A}^{\inf}$) est aussi non-déterministe.

La construction est extrêmement simple, il suffit de partir de l'automate, et de l'exécuter de manière concurrente sur toutes les branches.

Remarque 8.9. En général, si l'automate sur les mots n'est pas déterministe en histoire, cette construction est incorrecte. Illustrons cela pour les langages usuels. Dans ce cadre, la construction ci-dessus signifie que si l'on dispose d'un automate déterministe en histoire pour un langage L, alors le langage des arbres dont toutes les branches appartiennent à L est régulier. Considérons l'automate $\mathcal{A}$ qui accepte tous les mots non-vides sur l'alphabet $\{a, b\}$ qui a la structure suivante. Cet automate devine lors de la première transition si la dernière lettre du mot est un a ou un b, et vérifie à la fin de la lecture du mot que ce choix était le bon. Si l'on exécute un tel automate sur les branches de l'arbre, il effectuera le choix non-déterministe une fois à la racine de l'arbre. Ainsi, transformé en automate d'arbre, il

accepte le langage des arbres dont toutes les feuilles sont étiquetées par a, ou toutes les feuilles sont étiquetées par b. Ce n'est bien sûr pas le langage souhaité qui était simplement le langage de tous les arbres finis.

Ce problème est usuellement résolu en choisissant l'automate de départ déterministe. Dans le cas des automates à coût, ce n'est pas possible (les automates à coût déterministe étant strictement moins expressifs). Là encore, les automates déterministes en histoire permettent de contourner cette difficulté.

8.3 Décidabilité de la domination

Le cœur de toutes les procédures de décision sur les fonctions de coût correspond à la possibilité, étant données deux fonctions de coût sur les arbres f, g, de déterminer si la première domine la seconde, *c.-à-d.* $f \preccurlyeq g$. Sur les fonctions caractéristiques de deux langages K, L, rappelons que $\chi_K \preccurlyeq \chi_L$ est satisfait si et seulement si $L \subseteq K$ (remarque 2.17). Ainsi, décider $f \preccurlyeq g$ revient, dans le cas des langages, à décider d'une inclusion. Cette relation nous guide dans la suite de cette section.

Rappelons comment l'inclusion entre deux langages réguliers d'arbres infinis est effectuée en théorie classique des langages réguliers. La procédure usuelle pour tester $L \subseteq K$ consiste à complémenter K, obtenant ainsi un automate pour $K^{\complement}$, puis de construire un automate pour l'intersection $L \cap K^{\complement}$, et enfin de tester le vide de $L \cap K^{\complement}$. Bien entendu, ce langage est vide si et seulement si $L \subseteq K$. Ainsi, tester l'inclusion entre deux langages implique la combinaison de trois constructions (1) le complément, (2) l'intersection et, (3) le test du vide. Ce que nous pouvons faire pour les fonctions de coût suit le même schéma. Nous présentons donc ce que ces trois constructions signifient dans le cadre classique en même temps que nous précisons comment les adapter au cas des fonctions de coût.

(1) Le *complément* est un résultat difficile. Dans le cas des langages réguliers d'arbres, il s'agit du fameux lemme du complément de Rabin. Dans la théorie des fonctions régulières de coût, le complément entre automates non-déterministes correspond au résultat de dualité : être capable de transformer un automate non-déterministe avec objectif à minimiser en un automate non déterministe avec objectif à maximiser. En anticipant quelque peu sur le chapitre suivant, mentionnons que ce résultat, dans toute sa généralité, *c.-à-d.* la conversion effective entre $\mathtt{B} \wedge \mathtt{parité}$-automates non-déterministes d'arbres et $\mathtt{S} \wedge \mathtt{parité}$-automates non-déterministes d'arbres, est le problème ouvert principal concernant les fonctions régulières de coût. Nous verrons au cours de la section suivante comment traiter le cas des arbres finis. Dans cette section, nous esquivons cette difficulté, et nous supposons que les automates sont directement sous une forme permettant le déroulement de la suite de la procédure. Cela revient à supposer pour les langages que, par défaut, nous possédons un automate non-déterministe pour L et un automate non-déterministe pour $K^{\complement}$ (il s'agit après tout d'un codage de K comme un autre).

Dans notre cas, cela signifie que nous supposons que la fonction f est décrite au moyen

d'un automate non-déterministe $\mathcal{A}'$ d'objectif à maximiser O' (typiquement un S-automate), et g au moyen d'un automate non-déterministe $\mathcal{A}$ d'objectif à minimiser O (typiquement un B-automate). Tout autre choix nécessiterait de faire usage d'un résultat de dualité ou de simulation (*c.-à-d.* de «désalternation»), ce que nous cherchons à éviter.

(2) L'*intersection* consiste à effectuer le produit des deux automates, ce qui produit un automate dont la condition de gain est l'intersection des conditions de gain. Il s'agit de la construction effectuée dans le lemme 8.6. La subtilité ici provient du fait que nous cherchons à effectuer le produit d'un automate avec objectif à minimiser avec un automate avec objectif à maximiser. Il serait possible de définir de tels automates «hybrides» (voir, par exemple, le chapitre 12 dans [59]), et c'est ce qu'il est naturel de faire dans un cadre plus général (voir la piste 11.3). Nous esquivons de nouveau ce point technique en ne construisant pas effectivement cet automate, et en passant directement au «test du vide.»

(3) Le *test du vide* pour les automates non-déterministes s'interprète naturellement comme un jeu. Ainsi, étant donné un automate non-déterministe d'arbres infinis avec condition de gain W, il s'agit de résoudre le jeu suivant (nous supposons pour plus de simplicité qu'il n'y a pas de feuilles) :

- Les positions du jeu sont les états de l'automate. La position initiale du jeu est l'état initial de l'automate.
- À chaque tour de jeu, Ève choisit une transition de l'automate

$$(p, a, c_1, q_1, \ldots, c_k, q_k) \ ,$$

  telle que p est la position courante.
- Adam répond par un entier i entre 1 et k (une direction de sortie de a), et le jeu continue à partir de l'état q_i, en ayant vu la lettre c_i.
- Il s'agit d'un W-jeu, *c.-à-d.* que Ève gagne une partie si et seulement si la séquence des c_i's joués successivement appartient à W.

Ainsi, le jeu est une succession infinie de choix d'une transition de la part d'Ève et du choix d'une direction pour Adam. Ce jeu est gagné par Ève si et seulement si le langage accepté par l'automate est non-vide. En effet, si Ève gagne ce jeu, alors la stratégie gagnante témoignant de cette victoire est essentiellement un arbre de transitions puisque le seul choix de Ève est de choisir la transition suivante. Un tel arbre peut se voir comme une exécution acceptante de l'automate sur un arbre. La réciproque est similaire : étant donné un arbre accepté, l'exécution acceptante de l'automate est un arbre de transitions qui se transforme naturellement en une stratégie gagnante pour Ève dans le jeu ci-dessus.

Nous réutilisons exactement cette construction. Cette fois-ci, au lieu de chercher un témoin de non-vacuité, nous cherchons à produire un témoin de non domination pour $f \preccurlyeq g$. Ainsi, il s'agit de trouver un entier m tel que pour tout n il existe un arbre t tel que $g(t) \leq m$ et $f(t) > n$. En effet, un tel ensemble T d'arbres satisfait que $g(T)$ est borné, alors que $f(T)$ ne l'est pas. Il s'agit exactement du type de conditions de gain manipulées par les jeu de domination.

Rappelons que nous partons de deux automates non-déterministes $\mathcal{A}$ et $\mathcal{A}'$ respectivement d'objectif à minimiser O et à maximiser O', et qui acceptent les fonctions g et f respectivement. Nous effectuons le produit en même temps que nous construisons le jeu. Le jeu de domination résultat a la structure suivante :

- (phase spécifique aux jeux de domination) Ève choisit un entier m, et Adam répond par un entier n.
- Les positions sont des paires (p, p'), où p est un état de $\mathcal{A}$ et p' un état de $\mathcal{A}'$. La position initiale est formée de la paire des états initiaux de $\mathcal{A}$ et $\mathcal{A}'$ respectivement.
- À chaque tour de jeu, Ève choisit une paire de transitions

$$(p, a, c_1, q_1, \ldots, c_k, q_k),\ (p', a, c'_1, q'_1, \ldots, c'_k, q'_k)$$

  appartenant aux automates $\mathcal{A}$ et $\mathcal{A}'$ respectivement, et correspondant à une même lettre a (ici d'arité k).
- Adam répond en choisissant un entier i entre 1 et k. Le jeu continue à partir de la position (q_i, q'_i) en ayant vu la lettre (c_i, c'_i).
- L'objectif du jeu est $O_m \wedge O'_n$ (où O est interprété sur la première composante, et O' sur la seconde).

Ce jeu de domination est gagné par Ève si et seulement si $f \not\preccurlyeq g$. La preuve est similaire au cas du test du vide pour les langages réguliers.

La validité de cette construction est le contenu de la proposition suivante.

Proposition 8.10. *Étant donné un automate $\mathcal{A}$ non-déterministe d'objectif à minimiser O, et un automate non-déterministe $\mathcal{A}'$ d'objectif à maximiser O' le problème de déterminer si $[\![\mathcal{A}]\!] \preccurlyeq [\![\mathcal{B}]\!]$ se réduit à décider le vainqueur d'un jeu de domination fini d'objectif $O \wedge O'$.*

Bien entendu, ce qui nous intéresse est d'appliquer ces techniques pour décider de la domination entre fonctions de coût. Cela nous donne :

Corollaire 8.11. *Si f et g sont des fonctions de coût acceptées par respectivement un* `S` $\wedge$ `Muller-` *et un* `B` $\wedge$ `Muller`*-automate non-déterministe d'arbre infini, alors $f \preccurlyeq g$ est décidable.*

Démonstration. Il suffit de transformer, grâce à la proposition précédente les deux automates en un jeu de domination, puis d'appliquer le corollaire 7.44 pour en déterminer le vainqueur. CQFD

8.4 Le cas des arbres finis et la simulation

Nous sommes maintenant prêts à établir la décidabilité de la domination entre fonctions de coût définissables en logique monadique de coût sur les arbres finis.

Théorème 8.12. *La domination entre fonctions de coût définissables en logique monadique de coût sur les arbres finis est décidable.*

Là encore, nous suivons le schéma de preuve décrit à la section 2.6. Ainsi, notre objectif est de produire une classe de fonctions de coût représentable de manière finie, qui soit effectivement close sous min, max, inf-projection, sup-projection, qui contiennent certaines constantes, et sur laquelle la domination est décidable.

À ce stade de l'exposition, nous disposons pour l'instant sur les arbres finis des résultats suivants :

- La classe des fonctions *acceptées par* B*-automates non-déterministes* est close sous min, max et inf-projections (lemme 8.6). De plus, elle contient la fonction de coût qui compte le nombre d'occurrences d'une lettre (exemple 8.4), et les fonctions caractéristiques de tout langage régulier (il suffit de voir l'automate comme un B-automate sans compteurs pour reconnaître la fonction caractéristique).
- La classe des fonctions *acceptées par* S*-automates non-déterministes* est closes sous min, max et sup-projections (lemme 8.6),
- Si f est acceptée par un S*-automate d'arbre fini non-déterministe* et g par un B*-automate d'arbre fini non-déterministe*, alors $f \preccurlyeq g$ est décidable (corollaire 8.11).

Il nous suffit donc de montrer que les B-automates non-déterministes d'arbres finis sont effectivement $\approx$-équivalents aux S-automates non-déterministes d'arbres finis pour établir le théorème 8.12. Le sujet de cette section est d'énoncer et d'établir cette équivalence dans le cadre plus général des automates alternants.

Théorème 8.13. *Les familles d'automates suivantes définissent effectivement les mêmes fonctions de coût sur les arbres finis :*

- *les* hB*-automates non-déterministes,*
- *les* ¬hB*-automates alternants,*
- *les* B*-automates alternants,*
- *les* S*-automates non-déterministes,*
- *les* S*-automates alternants.*

Remarquons tout d'abord que les B-automates alternants et les ¬B-automates alternants sont équivalents sur les mots finis. Cela provient simplement du fait que, sur les mots finis, les conditions $\overline{\mathtt{B}}$ et ¬B sont équivalentes. En composant ensuite avec l'automate de la proposition 7.30 ces automates peuvent être transformés en hB-automates alternants ou en ¬hB-automates alternants.

Remarquons ensuite que, grâce au lemme 8.7 appliqué avec l'automate de la proposition 7.29 (un ¬B-automate universel pour S), tout S-automate alternant peut être transformé en un ¬B-automate alternant.

Ainsi, tous les types d'automates énoncés dans le théorème précédent peuvent donc être transformés, à la fois en hB-automates alternants ou en ¬hB-automates alternants. Le lemme de simulation suivant complète la boucle.

Lemme 8.14 (de simulation, [33])**.** *Tout* hB*-automate alternant d'arbre est effectivement équivalent à un* hB*-automate d'arbre non-déterministe. Tout* ¬hB*-automate alternant d'arbre est effectivement équivalent à un* S*-automate d'arbre non-déterministe.*

Remarque 8.15. Le terme de **simulation** provient des travaux de Muller et Schupp qui qualifient de simulation l'opération consistant à transformer un automate alternant en automate non-déterministe équivalent sur les arbres infinis [77]. De manière générale, les résultats de simulation sont plus forts que les résultats de complémentation. En effet, supposons que nous disposions d'un résultat de simulation, et considérons un automate non-déterministe de condition W. Cette automate peut toujours être vu comme un automate alternant de condition W. Ce dernier peut être transformé syntaxiquement en un automate alternant de condition $\overline{W}$ pour le langage complément. Enfin, au moyen d'une simulation, cet automate peut être rendu non-déterministe.

En pratique, dans le cas des langages réguliers, les preuves des résultats de simulation et de complément sont quasiment identiques. Dans le cas des automates d'arbres finis, il s'agit de constructions des parties. Les preuves de simulation et de complément dans le cas des arbres infinis s'expliquent au moyen de la théorie des jeux (voir par exemple [101], qui reprend les idées de [40]). Nous reprenons cette approche en terme de jeux dans le cas des fonctions de coût, même pour les arbres finis. En effet, les constructions des parties ne sont jamais suffisantes pour traiter les automates avec compteurs, tout comme elles sont insuffisantes pour les langages de mots et d'arbres infinis.

Revenons à la preuve du lemme 8.14. Nous partons d'un $\mathtt{hB}$-automate alternant d'arbre $\mathcal{A}$, et notre objectif est de le transformer en un $\mathtt{hB}$-automate non-déterministe d'arbre $\mathcal{B}$ qui accepte la même fonction de coût (la transformation d'un $\neg\mathtt{hB}$-automate alternant en un $\mathtt{S}$-automate non-déterministe procède des mêmes idées). Supposons que l'automate $\mathcal{A}$ possède comme ensemble d'états Q, comme état initial q_0, et comme fonction de transition δ.

Nous commençons par décrire comment une stratégie positionnelle σ_{E} dans le jeu $\mathcal{G}(\mathcal{A}, t)$ peut être codée au moyen d'une annotation de l'arbre t, *c.-à-d.* par un enrichissement des lettres à chaque nœud. Ainsi, un arbre sur l'alphabet gradué $\mathbb{A}$ annoté par une stratégie positionnelle pour Ève peut être vu comme un arbre sur l'alphabet

$$\mathbb{A}_\sigma = \{(a,S) \in \mathbb{A} \times (\{1,\dots,k\} \times \mathbb{C}_1 \times Q \ \cup\ \{\bot\} \times \mathbb{C}_0 \times \{\bot\})^Q \ : \\ \bigwedge S(p) \Rightarrow \delta(p,a) \text{ pour tout } p \in Q\} \ ,$$

où, naturellement, la lettre (a,S) a la même arité que a, c'est à dire k. Un arbre sur cet alphabet, décrit deux objets simultanément. D'un part, projeté sur sa première composante, il correspond naturellement à un arbre sur l'alphabet $\mathbb{A}$. D'autre part, sa deuxième composante induit une stratégie positionnelle σ_{E} comme suit. La racine de la stratégie est (ε, q_0), et elle est telle que pour tout nœud x portant la lettre (a,S) d'arité k, tout état p, et toute partie partielle $\pi \in \mathit{pref}(\sigma_{\mathrm{E}})$ se terminant en (x,p),

$$\pi((x,p),c,(xi,q)) \in \mathit{pref}(\sigma_{\mathrm{E}}) \quad \text{si et seulement si} \quad (i,c,q) \in S(p) \ .$$

Le choix de $\mathbb{A}_\sigma$, et en particulier l'hypothèse $\bigwedge S(p) \Rightarrow \delta(p,a)$, nous garantit qu'il s'agit bien d'une stratégie pour Ève dans le jeu $\mathcal{G}(\mathcal{A}, t)$. Nous noterons cet arbre simplement (t, σ_{E}) comme s'il s'agissait de la paire d'un arbre et d'une stratégie.

Ce qu'il nous faut maintenant, c'est un B-automate non-déterministe qui étant donné (t, σ_{E}) calcule la valeur de la stratégie σ_{E} dans le jeu $\mathcal{G}(\mathcal{A}, t)$. Ainsi, il s'agit de calculer le supremum sur toutes les stratégies π compatibles avec σ_{E} de $co\hat{u}t_{\mathtt{hB}}(\pi)$.

Remarquons tout d'abord ($\star$) que la condition hB a la propriété suivante, pour tout mot u sur $\mathbb{C}$:

$$co\hat{u}t_{\mathtt{hB}}(u) = \sup\{co\hat{u}t_{\mathtt{hB}}(v0_{\mathtt{hB}}) \ : \ v \text{ préfixe fini de } u\} \ ,$$

où $0_{\mathtt{hB}}$ dénote le terminateur 0 de la condition hB (ici, $v0_{\mathtt{hB}}$ dénote v lui-même si v se termine déjà par un terminateur). Cela provient en particulier du fait que pour n fixé, le langage $\{\pi \ : \ co\hat{u}t_{\mathtt{hB}}(u) \leq n\}$ est topologiquement clos. Considérons donc la fonction f qui à chaque mot de branche

$$(a_0, S_0)k_1(a_1, S_1)k_2 \ldots$$

associe

$$\sup\{co\hat{u}t_{\mathtt{hB}}(c_1c_2 \ldots c_\ell 0_{\mathtt{hB}}) \ : \ \text{il existe } q_1, q_2, \ldots, q_\ell, \\ \text{tels que pour tout } i = 1, \ldots, \ell, \ (k_i, c_i, q_i) \in S_i(q_{i-1})\} \ .$$

Cette fonction s'exprime en logique monadique de coût assez aisément. En fait, il est également facile de montrer qu'elle se décrit au moyen d'un hB-automate de mots universel dont les états sont essentiellement les états de l'automate $\mathcal{A}$ (certains états temporaires doivent néanmoins être utilisés pour traiter les directions $k_1, k_2, \ldots$ apparaissant dans le mot de branche). En tout état de cause, f ainsi définie appartient effectivement à une fonction régulière de coût sur les mots infinis. Il existe donc, d'après le corollaire 7.33 un hB-automate de mots non-déterministe qui l'accepte. Il s'ensuit d'après le lemme 8.8 qu'il existe un hB-automate d'arbre non-déterministe qui, étant donné un arbre (t, σ_{E}) étiqueté par une stratégie positionnelle, calcule (à $\approx$-près) le suprémum sur toutes les branches u de (t, σ_{E}) de $f(u)$, *c.-à-d.*, d'après les remarques précédentes, le coût de la stratégie σ_{E}. Soit F la fonction de coût ainsi définie. En utilisant le fait que les hB-jeux sont $\approx$-positionnellement déterminés (théorème 7.12), nous obtenons :

$$[\![\mathcal{A}]\!] \quad \approx \quad t \mapsto \inf\{F(t, \sigma_{\mathrm{E}}) \ : \ \sigma_{\mathrm{E}} \text{ stratégie positionnelle dans } \mathcal{G}(\mathcal{A}, t)\} \ .$$

Comme F est acceptée par un hB-automate d'arbre non-déterministe, et que les hB-automates d'arbres non-déterministes sont clos sous inf-projection (lemme 8.7), il s'ensuit que $[\![\mathcal{A}]\!]$ est effectivement reconnue par un hB-automate non-déterministe d'arbre.

Ainsi, nous avons montré comment un hB-automate alternant peut être effectivement transformé en un hB-automate non-déterministe d'arbre équivalent. Il s'agit du premier résultat énoncé par le lemme 8.14. Le second résultat, permettant de passer d'un $\neg$hB-automate alternant d'arbre à un S-automate d'arbre non-déterministe, s'obtient par un argument similaire.

Chapitre 9

Les arbres infinis

Ce chapitre présente brièvement les résultats connus concernant les fonctions régulières de coût sur les arbres infinis. En guise de décidabilité, toutes les questions sont ouvertes ou presque. En fait, toutes les techniques de décidabilité ont été vues au cours du chapitre précédent. Dans une première section (section 9.1), nous décrivons l'état des connaissances concernant la décidabilité sur les arbres infinis. Dans un second temps (section 9.2), nous présentons le problème de la hiérarchie de Mostowski, qui serait résolu si la logique monadique de coût était décidable sur les arbres infinis. Les arbres infinis seront de nouveau un sujet d'étude du chapitre suivant, le chapitre 10, qui traite de la logique monadique de coût faible.

9.1 Le problème ouvert de la décidabilité sur les arbres infinis

Comme nous l'avons déjà mentionné à de multiples reprises, le problème de décider la logique monadique de coût sur les arbres infinis est un problème qui reste ouvert.

Conjecture 9.1. *La domination entre fonctions de coût définissables en logique monadique de coût sur les arbres infinis est décidable.*

Pourtant, beaucoup de choses sont déjà connues, et la conjecture est que l'approche par automates développée dans le cadre des arbres finis reste valable, et en particulier le résultat de dualité.

Conjecture 9.2. *Les* $\mathtt{B} \wedge \mathtt{parité}$*-automates non-déterministes d'arbres infinis et les* $\mathtt{S} \wedge \mathtt{parité}$*-automates non-déterministes d'arbres infinis définissent effectivement les mêmes fonctions de coût.*

En effet, si la conjecture 9.2 s'avère établie, comme les fonctions de coût reconnues par les $\mathtt{B} \wedge \mathtt{parité}$-automates non-déterministes d'arbres infinis sont clos sous min, max, inf-projection et que celles reconnues par les $\mathtt{S} \wedge \mathtt{parité}$-automates non-déterministes d'arbres

infinis sont close sous sup-projection, cela signifierait que la classe de fonctions définies par ces automates est effectivement closes sous min, max, inf-projection et sup-projection. Comme de plus cette classe contient les constantes nécessaires, et que la question de la domination est décidable (corollaire 8.11), toutes les conditions sont réunies pour appliquer le fait 2.22 et en déduire que la domination entre formules de la logique monadique de coût est décidable sur les arbres infinis. Ainsi, la conjecture 9.2 implique la conjecture 9.1.

En fait, il est possible de pointer avec encore plus de précision où se trouve la difficulté. Si l'on cherche à démontrer la conjecture 9.2, tout comme dans le cas des arbres finis, le point crucial est l'existence de stratégies gagnantes à mémoire finie. Il suffirait d'établir la conjecture suivante.

Conjecture 9.3. *Les objectifs* $\texttt{hB} \wedge \texttt{parité}$ *et* $\neg\texttt{B} \wedge \texttt{parité}$ *sont à* $\approx$*-mémoire finie, sur toutes les arènes/sur les arènes chronologiques/sur les arènes «arborescentes».*

Cette conjecture vient en plusieurs variantes suivant que les arènes sont quelconques, chronologiques ou «arborescentes». Les arènes **arborescentes** sont tout simplement celles qui s'obtiennent par application d'un automate alternant fixé sur un arbre. En particulier les arènes arborescentes sont chronologiques, de degré borné, et de largeur arborescente bornée. Précisons aussi la dépendance entre les différents paramètres. Dans les conjectures ci-dessus, la mémoire nécessaire dans le jeu peut dépendre du nombre de compteurs, du nombre de priorités, et dans le cas arborescent, du nombre d'états de l'automate.

Établir l'$\approx$-mémoire finie pour les arènes quelconques est plus général que l'établir sur les arènes chronologiques, qui elles-même sont plus générale que les arènes arborescentes. Le dernier, et plus simple, de ces cas nous suffirait pour établir conjecture 9.2. Ce que nous essayons de souligner ici, en présentant trois variantes de cette conjecture, c'est qu'*a priori*, une preuve faisant appel à la propriété d'arborescence de l'arène aurait vraisemblablement une structure très différente de celle d'une preuve pour les arènes chronologiques. Il se peut également que la conjecture soit fausse dans le cas général, mais soit vraie pour les arènes arborescentes.

Le seul cas que l'on connaisse pour l'instant est celui des objectifs $\texttt{hB} \wedge \texttt{Büchi}$ et $\neg\texttt{hB} \wedge \texttt{Büchi}$, pour lesquels, sur les arènes chronologique, une $\approx$-mémoire 2 est suffisante (théorème 7.13). En appliquant ce résultat et une construction similaire à celle sur les arbres finis, nous obtenons le théorème suivant.

Théorème 9.4 (Vanden Boom [12])**.** *Les* $\texttt{hB} \wedge \texttt{Büchi}$*-automates alternants d'arbres infinis sont effectivement équivalents à leur version non-déterministe. Les* $\neg\texttt{hB} \wedge \texttt{Büchi}$*-automates alternants d'arbres infinis sont effectivement équivalents aux* $\texttt{S} \wedge \texttt{Büchi}$*-automates non-déterministes.*

Cette construction ressemble à la «breakpoint construction» qui permet, sur les mots, (a) de transformer un co-Büchi-automate non-déterministe en un co-Büchi-automate déterministe [73], et (b) de transformer un Büchi-automate alternant d'arbres infinis en un

Büchi-automate non-déterministe [75]. Cette construction repose sur les propriétés particulières de la condition Büchi et fait en particulier appel au lemme de König. Il s'agit d'une situation qui n'est en rien représentative du cas général. La situation est similaire ici. Certes le cas Büchi est traitable, mais les techniques impliquées pour cela, et en particulier l'usage du lemme de König comme dans la preuve du théorème 7.13, ne sont pas généralisables.

Un autre cas intéressant traite des automates bidirectionnels (*two-way*). Ces automates ont la possibilité de remonter dans l'arbre. Nous ne formalisons pas plus ce modèle de contrôle qui est classique.

Théorème 9.5 (non publié, avec C. Löding)**.** *Les* distance ∧ parité*-automates alternants bi-directionnels peuvent être effectivement transformés en* B ∧ parité*-automates non-déterministes équivalents et en* S ∧ parité*-automates non-déterministes équivalents.*

Le point crucial de ce résultat est bien entendu la proposition 7.14 qui énonce l'existence de stratégies sans mémoire dans les jeux correspondants. Le reste de la construction est un peu différente de ce qui est expliqué ci-dessus en raison de la nature bi-directionnelle de la stratégie.

Dans le cas plus général d'un objectif hB ∧ parité, la situation devient beaucoup plus complexe. Le problème est l'interférence entre les priorités impaires, et les remises à zéro des compteurs. En particulier, un problème typique est de départager la situation suivante, dans un jeu de condition B ∧ co-Büchi à un compteur :

> Vaut-il mieux (a) voir une priorité 0 et aller dans un état, ou bien (b) remettre à zéro le compteur, voir la priorité 1 et aller dans un autre état ?

Il y a un compromis à faire. Si l'on choisit trop souvent (a), alors il y a le risque d'incrémenter trop le compteur, alors que si l'on choisit (b), il y a le risque de produire une partie contenant une infinité de 1. Dans les deux cas, la stratégie est perdante. À priori, il faudrait utiliser la valeur courante du compteur pour effectuer ce choix de manière optimale. Malheureusement une stratégie à $\approx$-mémoire finie ne peut accéder à cette information. Cette situation type est au centre de nombreux contre-exemples mettant en échec diverses tentatives de preuves.

9.2 La hiérarchie de Mostowski

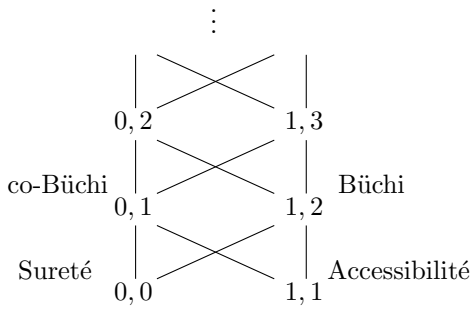
L'une des motivations pour l'étude de la logique monadique de coût sur les arbres infinis, outre son intérêt intrinsèque et le fait que cela serait la première extension du résultat de Rabin à une logique plus expressive que la logique monadique, provient d'une réduction la mettant en lien avec la hiérarchie de Mostowski. Nous présentons ce résultat au cours de cette section.

La **hiérarchie de Mostowski** (ou de point-fixe, ou de Rabin) s'énonce en de nombreuses variantes. En effet, elle s'applique à toute forme d'automates possédant des exécutions infinies. En particulier, elle concerne les automates déterministes, non-déterministes ou alternants sur les mots infinis et les arbres infinis. Dans chacune de ces combinaisons

la hiérarchie possède une structure différente. Mostowski s'est intéressé à cette indice dans [74].

La façon la plus simple d'introduire cette hiérarchie est au moyen de la condition d'acceptation de parité que nous avons déjà rencontrée à de nombreuses reprises. Chacun des modèles d'automates décrits ci-dessus (déterministes, non-déterministes, alternants, ...) peut être équipé de différentes formes de conditions d'acceptations, et en particulier de la condition de parité. La condition de parité est paramétrée par un intervalle d'entiers $[i, j]$ appelés priorités, la condition étant remplie quand la plus grande priorité rencontrée infiniment souvent est paire. Un automate est dit d'**indice** (de Mostowski) i, j s'il utilise uniquement des priorités dans l'intervalle $[i, j]$. Un langage (de mot, ou d'arbre, etc...) est dit i, j**-faisable** (par rapport à une certaine famille d'automates) s'il existe un automate de cette famille, d'indice i, j, qui reconnaît ce langage.

Il est aisé de vérifier que décaler toutes les priorités d'un automate de parité de $+2$ ou de -2, transforme l'automate en un automate équivalent. En effet, la parité de la plus grande priorité apparaissant infiniment souvent reste inchangée. Cela signifie que nous pouvons toujours nous ramener à des intervalles $[i, j]$ pour lesquels $i = 0$ ou $i = 1$. Remarquons de plus que si $[i, j] \subseteq [i', j']$ alors tout automate d'indice i, j peut être également vu comme un automate d'indice i', j'. Ainsi, en général, la hiérarchie à la structure suivante.



Remarquons que dans le cas de l'indice $1, 1$ (accessibilité), pour les automates non-déterministes, seuls des mots ou des arbres finis peuvent être acceptés. Ainsi, si nous ne considérons que des mots réellement infinis (de longueur ω), ou des arbres binaires infinis complets (qui n'ont pas de feuilles), alors cette classe est pathologiquement vide.

Remarquons aussi que le dual (c'est à dire son complémentaire ici) d'une condition de parité d'indice i, j est une condition de parité d'indice $i + 1, j + 1$ (à échange des terminateurs près). Cela signifie que pour les classes d'automates pour lesquelles le contrôle peut être dualisé (*c.-à-d.* les automates déterministes et complets et les automates alternants, mais pas les automates non-déterministes), les langages i, j-faisables sont exactement les complémentaires des langages $i + 1, j + 1$-faisables.

Remarque 9.6. Remarquons enfin qu'un langage n'a pas forcément un unique indice. Par exemple, sur l'alphabet $\{a, b\}$, considérons le langage de mots infinis L suivant :

> «si la première lettre est un a, il y a une infinité d'occurrences de la lettre a, sinon, il n'y a qu'un nombre fini d'occurrences de la lettre b.»

Étudions la faisabilité de ce langage dans la hiérarchie des automates déterministes de mots infinis. Il se trouve que le langage «il y a une infinité de a» est $1,2$ (*c.-à-d.* `Büchi`), et d'aucun niveau inférieur. De même, le langage «il y a un nombre fini de b» est $0,1$ (*c.-à-d.* `co-Büchi`), et d'aucun niveau inférieur. Le langage L est plus complexe que ces deux cas (ce qui peut se montrer par des techniques de réduction). Il doit donc être au dessus de $0,1$ et de $1,2$. Pourtant, il est aisé de montrer que L est à la fois $0,2$ et $1,3$-faisable. Il ne possède donc pas d'unique indice minimal dans la hiérarchie : il est à la fois $0,2$ et $1,3$.

L'intérêt de ce paramètre est multiple. Tout d'abord, il s'agit d'un paramètre important pour la complexité de différents algorithmes sur les automates. En effet, tester le vide d'un automate non-déterministe d'arbres infinis de parité est un problème connu comme appartenant à NP∩coNP, mais qui n'est pas connu comme appartenant à P (ce problème est équivalent à la résolution des jeux de parité, et savoir si ce problème est dans P est l'un des grands problèmes ouverts en théorie des automates). En revanche, si l'indice de l'automate est fixé, alors ce problème est tout naturellement dans P. Une autre raison provient du fait que la condition de parité correspond à l'imbrication d'alternances des points-fixes quand le langage est exprimé en μ-calcul. Un plus petit point-fixe μ correspondant à une priorité impaire, et un plus grand point-fixe ν à une priorité paire. Ainsi, étudier la hiérarchie de Mostowski, ce n'est rien d'autre que d'étudier la hiérarchie induite par l'alternance des points-fixes en μ-calcul.

Présentons enfin les résultats connus concernant l'indice de Mostowski, suivant la classe d'automates considérés.

Automates déterministes de mots infinis. Dans ce cas, la hiérarchie est stricte [104]. Le problème, étant donné un langage de mots infinis, de déterminer s'il est i,j-faisable est décidable (en temps polynomial si l'automate en entrée est un automate de parité déterministe [22]).

Automates non-déterministes de mots infinis. Il est connu que les langages réguliers de mots infinis s'expriment au moyen d'automates non-déterministes de Büchi. Cela signifie que la hiérarchie dans ce cas s'effondre au niveau $1,2$. Les classes $1,1$ et $0,0$ sont incomparables [1], et strictement plus faibles que le niveau $0,1$ (`co-Büchi`), lui même strictement plus faible que le niveau $1,2$ (`Büchi`). Mentionnons enfin que les niveaux $0,0$, $1,1$, et $0,1$ coïncident avec le niveau correspondant de la hiérarchie déterministe. Pour les niveaux $0,0$ et $1,1$, il s'agit d'une construction des parties. Pour le cas $0,1$, il s'agit de la «breakpoint construction» déjà mentionnée [73].

Automates alternants de mots finis. Cette fois-ci, la hiérarchie s'effondre encore plus bas, puisque tout langage régulier s'exprime à la fois comme un langage de `Büchi`, et comme un `co-Büchi` alternant. Ainsi, tout langage régulier est à la fois d'indice $1,2$ et

1. Si l'on accepte les mots finis comme cas particulier, sinon le niveau $1,1$ ne contient aucun langage, et est pathologiquement inclus dans le niveau $0,0$.

d'indice $0, 1$.

Automates non-déterministes d'arbres infinis. Dans ce cas, contrairement au cas des mots, la hiérarchie est stricte [81]. Le langage montrant cette séparation est là encore très simple. Étant donnés i, j, ce langage est sur l'alphabet $[i, j]$ et contient les arbres infinis tel que le long de toutes les branches, l'entier le plus grand apparaissant infiniment souvent est pair. Quelques résultats sont connus sur ce cas. En particulier, si L est un langage accepté par un automate déterministe descendant d'arbre (tous les langages réguliers d'arbres infinis ne peuvent être mis sous cette forme), alors il est possible de décider si ce langage est d'indice i, j dans la classe des automates non-déterministes [83, 82]. Soulignons qu'il faut être prudent en énonçant ce résultat : l'automate en entrée est déterministe, mais l'indice de Mostowski est étudié par rapport à la classe des automates non-déterministes.

Automates alternants d'arbres infinis. Là encore, la hiérarchie est stricte. Aucun niveau de cette hiérarchie n'est connu comme décidable. Notons tout de même que le niveau $1, 2$ (de Büchi) de cette hiérarchie est équivalent au niveau $1, 2$ de la hiérarchie non-déterministe (il s'agit de nouveau de la «breakpoint construction»).

L'une des motivations de l'étude des automates à coût sur les arbres infinis a été pour l'étude de l'indice de Mostowski pour les automates non-déterministes d'arbres infinis, et plus précisément dans l'objectif de décider du niveau d'un langage régulier donné.

Théorème 9.7 ([32])**.** *Déterminer si un langage régulier d'arbres infinis L est d'indice de Mostowski i, j dans les automates non-déterministes se réduit au problème de déterminer, étant donné un* $\mathtt{B} \wedge \mathtt{parité}$*-automate non-déterministe d'arbres infinis utilisant le même intervalle i, j de priorités, si la fonction qu'il calcule est bornée sur L.*

La réduction impliquée dans ce théorème dépasse largement le cadre de ce document. Elle met en jeu l'existence de formes spéciales d'automates à parité sur les arbres infinis, dits automates guidables : informellement, les automates guidables utilisent leur non-déterminisme avec suffisamment de souplesse pour pouvoir «simuler» le comportement de tout automate reconnaissant le même langage. Il s'agit d'une propriété qu'ont en particulier tous les automates déterministes descendants. Un important fait est que tout langage régulier d'arbres infinis est accepté par un automate guidable. La notion d'automate guidable est étudiée avec précision dans [67], ainsi que la preuve du théorème 9.7.

Remarque 9.8 (sur la généralité de l'approche). La réduction énoncée dans le théorème 9.7 possède une similarité formelle avec celle impliquée dans le problème de la hauteur d'étoile. En effet, tous deux sont de la forme :

> «Étant donnée une classe d'automates non-déterministes reconnaissant la classe de langages $\mathcal{C}$, est-ce qu'un langage donné K est dans $\mathcal{C}$?»

Si $\mathcal{C}$ est formé des langages de hauteur d'étoile k, il s'agit du problème de la hauteur d'étoile[2], sur les mots comme sur les arbres. Si $\mathcal{C}$ est formé des langages réguliers d'arbres

2. Les expressions régulières peuvent être vues, quasi syntaxiquement, comme des automates non-

infinis i, j-faisables, il s'agit du problème de décider du niveau dans la hiérarchie de Mostowski.

Cette analogie est à l'origine du théorème 9.7. Dans les deux cas la réponse prend la forme d'une réduction à un problème de la forme :

«Est-ce qu'une fonction calculée par un automate à coût est bornée sur K ?»

En fait, cette approche est en un sens très générale. Ainsi, considérons une famille de langages $\mathcal{C}$ (par exemple les langages reconnus par un automate à parité i, j, ou les langages reconnus par une expression de hauteur d'étoile k, etc...). Nous ne faisons que trois hypothèses sur $\mathcal{C}$: (1) à chaque langage $L \in \mathcal{C}$ est associée la taille n_L du plus petit automate qui le reconnaît, (2) pour tout n, $\{L \in \mathcal{C} \ : \ n_L = n\}$ est fini, et (3) la classe $\mathcal{C}$ est close sous union finie.

Considérons maintenant, pour un langage quelconque K, la fonction des entrées dans $\mathbb{N} \cup \{\infty\}$ suivante :

$$f_K(t) = \inf\{n_L \ : \ t \in L \subseteq K, \ L \in \mathcal{C}\} \ .$$

Alors, nous affirmons que

($\star$) f_K est bornée sur K si et seulement si $K \in \mathcal{C}$.

En effet, supposons $K \in \mathcal{C}$, alors clairement pour tout $t \in K$, $f_K(t) \leq n_K$ puisque $t \in K$ et $K \subseteq K$. Donc f_K est bornée sur K. Supposons maintenant que f_K soit bornée sur K, disons par n, et considérons le langage

$$K' = \bigcup_{L \in \mathcal{C}, \ L \subseteq K, \ n_L \leq n} L \ .$$

Bien entendu, K' est une union finie de langages de $\mathcal{C}$, et est donc un langage de $\mathcal{C}$. De plus il s'agit d'une union de langages $L \subseteq K$, et donc $K' \subseteq K$. Enfin, considérons $t \in K$, alors il exists $L_t \in \mathcal{C}$ tel que $n_{L_t} \leq n$ et $t \in L_t \subseteq K$. Le langage L_t apparaît donc dans l'union définissant le langage K', donc $t \in L_t \subseteq K$. Ainsi, l'affirmation ($\star$) est validée.

Ainsi, réduire un problème d'appartenance d'un langage à une classes se traduit naturellement en un problème d'existence de bornes. Et c'est en pratique ce qu'il se passe. Ainsi, pour la hauteur d'étoiles sur les mots, pour la hauteur d'étoiles sur les arbres, comme pour l'indice de Mostowski, la réduction s'obtient en produisant un automate de coût pour f_K, ou plus exactement pour la fonction de coût de f_K. Les techniques employées dans ce type de réduction sont ensuite spécifiques à chaque problème. À la connaissance de l'auteur, seules des classes de machines non-déterministes permettent d'effectuer ce type de réduction, bien que ce ne soit pas formellement requis par les explications ci-dessus.

déterministes possédant une structure particulière. Ce paramètre de structure, dit d'entrelacement, a été étudié par Eggan [37].

Comme le problème de décider de l'existence d'une borne pour les automates hB∧parité est pour l'instant ouvert, cette approche ne permet pas de conclure à la décidabilité de la hiérarchie non-déterministe de Mostowski sur les arbres infinis.

Conjecture 9.9. *Le problème, étant donné un langage régulier L d'arbres infinis, et deux entier i, j, de déterminer si L est accepté par un automate non-déterministe de parité i, j, est décidable.*

Bien entendu, une conséquence du théorème 9.7 est que si la conjecture 9.1 est vérifiée, alors la conjecture 9.9 l'est aussi.

Pourtant, cette approche a permis de donner deux nouveaux résultats partiels dans cette direction.

Théorème 9.10. *Le niveau* $0, 1$ *(*co-Büchi*) de la hiérarchie de Mostowski non-déterministe sur les arbres infinis est décidable.*

Démonstration. Soit f une fonction de coût calculée par un hB ∧ co-Büchi-automate alternant (seul le cas non-déterministe nous intéresse, mais nous ne nous servirons pas de cette hypothèse) d'arbres infinis, et L un langage régulier d'arbres infinis. Notre objectif est de décider si f est bornée sur L, c'est à dire si $f \preccurlyeq \chi_L$.

Le hB ∧ co-Büchi-automate se transforme effectivement, en le dualisant et en le composant avec l'automate déterministe en histoire de la proposition 7.31 en un ¬hB ∧ Büchi-automate alternant. Par le théorème 9.4, cet automate peut être transformé en un S∧Büchi-automate non-déterministe d'arbres infinis.

Le langage L, quant à lui, est accepté par un automate de parité, qui, vu comme un B∧parité-automate, calcule la fonction χ_L. Or f est bornée sur L si et seulement si $f \preccurlyeq \chi_L$. Ce dernier problème est décidable par le corollaire 8.11. CQFD

Le deuxième résultat, dû à Vanden Boom et Kuperberg, est plus subtil.

Théorème 9.11 (Vanden Boom et Kuperberg, non publié)**.** *Le problème de déterminer, étant donné un langage* $1, 2$*-faisable (*Büchi*), si son complémentaire est aussi* $1, 2$*-faisable (*Büchi*), est décidable.*

En fait, les langages à la fois Büchi et de complémentaire Büchi sont bien connus, il s'agit des langages faibles [88]. Nous aurons l'occasion de présenter ce résultat de Rabin plus en détail au chapitre suivant (théorème 10.4) qui traite de la logique monadique de coût faible. Ainsi, le théorème 9.11 peut-il être reformulé comme suit.

Théorème 9.12. *Le problème de déterminer, étant donné un langage* $1, 2$*-faisable (*Büchi*), s'il est faible, est décidable.*

Nous ne présentons pas cette preuve. Elle peut être trouvée dans la thèse de Kuperberg [59]. Elle est plus complexe que la preuve du théorème 9.10 et nécessite en particulier d'utiliser des résultats fins concernants la classe des automates à coût alternants faibles.

Avec les résultats de Niwiński et Walukiewicz [83, 82] qui traitent entièrement le cas des langages déterministes descendants en entrée, il s'agit des seuls résultats de décidabilité connus pour la hiérarchie de Mostowski non-déterministe sur les arbres infinis.

Chapitre 10

La logique monadique de coût faible et ses variantes

Ce chapitre traite de fragments de la logique monadique de coût sur les mots et les arbres infinis, le problème général restant ouvert (*cf.* conjecture 9.1). Les résultats présentés dans ce chapitre sont principalement dus à Michael Vanden Boom et Denis Kuperberg [12, 60, 13, 61].

La section 10.1 a pour but de rappeler les résultats classiques concernant la logique monadique faible, et de présenter la suite du chapitre. La section 10.2 traite de la logique monadique de coût faible. La section 10.3 introduit la classe plus expressive des automates alternants de Büchi compteur-faibles. Enfin, la section 10.4 traite de l'effondrement de toutes ces classes sur les mots infinis.

10.1 Logique monadique faible

La **logique monadique faible** a la même syntaxe que la logique monadique. La seule différence est que les variables monadiques ne sont pas interprétées comme des ensembles, mais comme des ensembles *finis*. Bien entendu, cela ne fait aucune différence sur les structures finies. Sur les arbres infinis, la logique monadique faible est strictement moins expressive que la logique monadique. En particulier, il est impossible d'exprimer des propriétés comme «il existe une branche contenant une infinité d'occurrences de la lettre a». En revanche, sur les mots infinis (de longueur ω), l'expressivité de ces deux logiques est identique.

Théorème 10.1 (McNaughton [70])**.** *La logique monadique et la logique monadique faible définissent effectivement les mêmes langages de mots infinis.*

Il s'agit d'une conséquence du théorème de determinisation sur les mots infinis. En effet, considérons un énoncé de la logique monadique. Grâce au théorème de McNaughton, il s'exprime au moyen d'un automate de Muller déterministe. Or, il est possible de produire

une formule de logique monadique faible exprimant que «à la position x, l'état de la seule exécution de l'automate sur ce mot est q» (c'est assez simple puisque cet énoncé ne dépend que du mot jusqu'à la position x, c'est à dire d'une portion finie du mot). Il est alors facile d'exprimer des propriétés comme «l'état p apparait une infinité de fois», et donc, par combinaison booléenne, le fait que l'unique exécution de l'automate de Muller déterministe est acceptante.

Remarque 10.2. La logique monadique faible n'est pas à proprement parler un fragment de la logique monadique. En effet, la logique monadique ne permet pas d'exprimer la propriété «être fini». Ainsi, sur une structure ne possédant que la prédicat unaire a, il est possible d'exprimer en logique monadique faible que seul un nombre fini d'éléments portent la propriété a (par exemple par la formule «il existe un ensemble (fini car il s'agit de la logique monadique faible) tel que tous les éléments ayant la propriété a en font parti»), alors que c'est impossible en logique monadique (sur une signature ne possédant que des relations unaires, la logique monadique ne peut qu'exprimer que des combinaisons booléennes de propriété de la forme «il y a exactement k éléments satisfaisant une combinaison booléenne de prédicats unaires»). Néanmoins, sur toutes les structures auxquelles nous nous intéressons dans ce document, la logique monadique faible peut être vue comme un fragment de la logique monadique. En effet, la propriété «être infini» pour un ensemble X peut être exprimée en logique monadique sur toute structure possédant un ordre total. Pour cela il suffit de deviner une partie de X qui, équipée de l'ordre total, est isomorphe à ω, ou son renversé ω^*. C'est bien entendu le cas pour les mots, c'est aussi indirectement le cas sur les arbres puisque l'ordre lexicographique sur les nœuds de l'arbre est définissable et total.

Sur les arbres infinis (ou les mots infinis), il se trouve qu'il est significativement plus simple (en termes conceptuels, et non pour des questions de complexité algorithmique) de décider de la satisfaction de la logique monadique faible que pour la logique monadique complète. En particulier, il n'est pas nécessaire d'utiliser de résultats avancés sur les jeux comme la détermination à mémoire finie. Une façon d'établir la décidabilité de la logique monadique faible sur les arbres infinis est de la traduire en une forme spéciale d'automate : les automates alternants faibles. Un automate (alternant) de Büchi sera dit **faible** si :

– Pour tout cycle, soit toutes les transitions sont Büchi, soit toute les transitions sont non-Büchi.

Le résultat de cette définition est que si une partie (sur le jeu induit par un arbre) est gagnante pour la condition de Büchi, alors à partir d'un moment, la composante fortement connexe finale étant atteinte, toutes les transitions sont de Büchi. Une autre façon de voir est que la condition de Büchi peut être vue dans ce cas, également comme une condition co-Büchi. En effet, sous l'hypothèse d'être faible, «voir une infinité de fois une transition de Büchi» est équivalent à «ne voir qu'un nombre fini de transitions non-Büchi». Pour cette raison, les automates de Büchi alternants faibles sont naturellement clos sous complément.

Les automates faibles sont en correspondance directe avec la logique monadique faible comme le montre le théorème suivant.

Théorème 10.3 (Muller, Saoudi, Schupp [75]). *Un langage d'arbres infinis s'exprime en logique monadique faible si et seulement s'il est effectivement accepté par un automate alternant faible.*

Les propriétés d'arbres exprimables en logique monadique faible possèdent une caractérisation élégante, due à Rabin.

Théorème 10.4 (Rabin [88]). *Un langage d'arbres infinis L est exprimable en logique monadique faible si et seulement si L et le complémentaire de L sont reconnus par des automates non-déterministes*[1] *de Büchi.*

La preuve originale de Rabin ne passe pas par les automates alternants faibles, et montre directement l'équivalence avec la logique monadique faible, ce qui est équivalent. Kupferman et Vardi ont donné une preuve simplifiée, sous la forme ci-dessus, de ce résultat [62]. Remarquons néanmoins que cette caractérisation ne permet pas de décider, étant donné un langage régulier d'arbres infinis, s'il est exprimable en logique monadique faible (nous dirons simplement qu'il est **faible**).

Dans ce contexte, il est naturel de considérer le fragment faible de la logique monadique de coût. La **logique monadique de coût faible** possède la même syntaxe que la logique monadique de coût, en revanche, lorsqu'une formule de cette logique est évaluée, les variables monadiques ne peuvent prendre comme valeur que des ensembles finis.

Plusieurs questions se posent alors naturellement.

1. Peut-on décider de la logique monadique de coût faible sur les arbres infinis ? En effet, la décidabilité de la logique monadique de coût sur les arbres infinis est un problème ouvert (conjecture 9.1). Nous verrons que le théorème 10.5 répond positivement à cette question, et pour cela établit une équivalence entre la logique monadique de coût faible et les $\mathtt{B} \wedge \mathtt{Büchi}$-automates alternants faibles.

2. La logique monadique de coût et sa variante faible ont-elles la même expressivité sur les arbres infinis ? La réponse à cette question est simple : non. Cela s'obtient par réduction au résultat de séparation pour les langages réguliers.

3. La logique monadique de coût et la logique monadique de coût faible ont-elles la même expressivité sur les mots infinis ? En effet, contrairement au cas des langages, les automates ne peuvent être déterminisés. Il n'est donc pas possible d'utiliser l'argument de déterminisation de McNaughton (théorème 10.1). Les deux logiques sont pourtant bien équivalentes, comme l'établira le théorème 10.10.

4. Peut-on caractériser la logique monadique de coût faible sur les arbres infinis comme les fonctions de coût qui sont reconnu à la fois par un $\mathtt{B} \wedge \mathtt{Büchi}$-automate non-déterministe et par un $\mathtt{S} \wedge \mathtt{Büchi}$-automate non-déterministe ? Il s'agirait de l'énoncé correspondant naturellement au théorème 10.4 dans l'univers des fonctions de coût. Nous verrons que la réponse à cette question est négative, et que les fonctions de coût qui sont à la fois reconnues

1. Ou, de manière équivalente, des automates alternants de Büchi, grâce à la «breakpoint construction» [78].

par $\mathtt{B} \wedge \mathtt{Büchi}$-automate non-déterministes et par $\mathtt{S} \wedge \mathtt{Büchi}$-automate non-déterministes forment une classe strictement plus grande que les fonctions de coût définissable en logique monadique de coût faible : la classe des fonctions de coût reconnues par les automates $\mathtt{B} \wedge \mathtt{Büchi}$-automates alternants compteur-faibles.
L'étude de ces questions étaient à l'origine de la thèse de Michael Vanden Boom, à Oxford, et a été l'objet de collaborations avec Denis Kuperberg, étudiant en thèse à Paris. À l'issu des ces travaux, toutes ces questions ont trouvé une réponse.

Dans la suite de ce chapitre, nous donnons quelques précisions supplémentaires sur ces résultats.

10.2 Logique monadique de coût faible sur les arbres infinis

La première question à laquelle il s'agissait de répondre était la décidabilité de la relation de domination pour les fonctions définies en logique monadique de coût faible sur les arbres infinis. Vanden Boom établit ce résultat en utilisant une approche similaire à celle de la logique monadique faible, c'est à dire en faisant usage d'automates alternants faibles. Ainsi, un $\mathtt{B} \wedge \mathtt{Büchi}$-automate alternant est dit **faible** si (la définition est identique à celle dans le cas des langages) :

– Pour tout cycle, soit toutes les transitions sont Büchi, soit toute les transitions sont non-Büchi.

Théorème 10.5 (Vanden Boom [12])**.** *Les énoncés suivants sont effectivement équivalents pour une fonction de coût sur les arbres infinis :*

– *être définissable en logique monadique de coût faible,*
– *être reconnue par un* $\mathtt{B} \wedge \mathtt{Büchi}$*-automate alternant faible,*
– *être reconnue par un* $\neg\mathtt{B} \wedge \mathtt{Büchi}$*-automate alternant faible.*

Ce résultat, combiné avec le théorème 9.4 (simulation pour les automates de coût $\mathtt{B} \wedge \mathtt{Büchi}$ et $\mathtt{S} \wedge \mathtt{Büchi}$) et le corollaire 8.11 (décidabilité de la domination entre fonctions acceptées par automates $\mathtt{B} \wedge \mathtt{parité}$ et $\mathtt{S} \wedge \mathtt{parité}$), nous donne alors le corollaire suivant.

Corollaire 10.6. *La logique monadique de coût faible est décidable sur les arbres infinis.*

En suivant l'approche déjà utilisée à de nombreuse reprise au cours de ce document, démontrer un tel résultat revient essentiellement à démontrer que les $\mathtt{B} \wedge \mathtt{Büchi}$-automates alternants faibles satisfont certaines propriétés de clôture effective.

Les clôtures sous min et max sont bien entendu évidentes, comme toujours pour les automates alternants. Le point intéressant est la clôture sous inf-projection et sup-projection. En fait, comme il s'agit de la variante faible de la logique, il s'agit de démonter un résultat plus faible de projection : en quelque sorte, la projection se fait par rapport à une partie finie de l'arbre (en effet, la projection correspond à deviner la valeur d'une variable monadique. Or cette variable monadique peut être circonscrite à un préfixe fini de l'arbre

dans le cas de la logique monadique faible). Rappelons que la clôture sous inf-projection est aisée sur un automate non-déterministe (lemme 8.6). Dans le cas présent, nous ne disposons que d'automates alternants. Il convient donc de «désalterner» (ce que l'on appelle la simulation, *cf.* lemme 8.14) l'automate alternant. Le problème que nous avons alors est que cette procédure produit un $\mathtt{B} \wedge \mathtt{Büchi}$-automate non-déterministe ou un $\mathtt{S} \wedge \mathtt{Büchi}$-automate non-déterministe équivalent. Or, ces formes d'automates sont strictement plus expressives que la logique monadique faible. L'astuce dans le cas des logiques monadiques faibles, et donc en particulier dans le cas de la logique monadique de coût faible, consiste à effectuer une *simulation partielle*, c'est à dire à ne transformer l'automate alternant en automate non-déterministe que sur une portion finie de l'arbre.

10.3 Les automates compteur-faibles et leur caractérisation

Nous avons rencontré le théorème 10.4 qui caractérise les langages d'arbres infinis définissables en logique monadique faible comme ceux qui sont acceptés par un automate non-déterministe de Büchi, et dont le complémentaire est aussi accepté par un automate non-déterministe de Büchi. Il est naturel de s'interroger sur une éventuelle extension de ce théorème au cas des fonctions de coût sur les arbres infinis. Une question naturelle est de déterminer tout d'abord quelle est la caractérisation «équivalente» à «être accepté par un automate de Büchi et avoir son complémentaire accepté par un automate de Büchi.» La question correspondante pour les fonctions de coût est la suivante :

> Quels sont les langages d'arbres infinis qui sont à la fois acceptés par un $\mathtt{B} \wedge \mathtt{Büchi}$-automate non-déterministe et par un $\mathtt{S} \wedge \mathtt{Büchi}$-automate non-déterministe ?
>
> En particulier, est-ce que cette classe coïncide avec les langages définissables en logique monadique faible ?

Si le résultat de Rabin s'étendait aux fonctions de coût, la réponse à la deuxième question devrait être positive. Kuperberg et Vanden Boom, qui ont étudié finement cette classe, ont répondu par la négative. Nous décrivons dans la suite de cette section leur contribution.

Leur réponse ne fait en pratique pas référence à la logique monadique de coût faible, mais utilise sa caractérisation en termes d'automates faibles (théorème 10.5). Ainsi, nous avons vu la notion de $\mathtt{B} \wedge \mathtt{Büchi}$-automates alternants de coût faibles, dont l'expressivité est équivalente à la définissabilité en logique monadique de coût faible. Dans de tels automates, le long de tout chemin (*c.-à-d*, dans toute partie du jeu obtenu en appliquant l'automate à un arbre), syntaxiquement, l'automate ne peut passer qu'un nombre borné de fois entre des transitions de Büchi et des transitions qui ne sont pas de Büchi. Cela s'exprime comme suit :

- tout cycle dans l'automate est composé, soit uniquement de transitions Büchi, soit uniquement de transitions non-Büchi.

Ainsi, passer d'une transition de Büchi à une transition non-Büchi (ou vice-versa), ne peut se faire qu'en changeant de composante fortement connexe.

Les $\mathtt{B} \wedge \mathtt{Büchi}$-automates alternants compteur-faibles [2] bornent également le nombre de changements possibles entre une transition Büchi et une transition non-Büchi, mais le font au moyen de leurs compteurs. Ainsi, un $\mathtt{B} \wedge \mathtt{Büchi}$-automate alternant est **compteur-faible** s'il satisfait la contrainte suivante :

- pour tout cycle dans l'automate qui contient à la fois une transition Büchi et une transition non-Büchi, il existe un compteur (de la condition $\mathtt{B}$) qui est incrémenté et n'est pas remis à zéro.

Ainsi, si au cours d'une partie dans le jeu induit par l'automate appliqué à un arbre, un grand nombre d'alternances entre des transitions Büchi et des transitions non-Büchi est rencontrée, alors inévitablement, l'un des compteurs prendra une grande valeur, et l'arbre ne sera pas accepté.

Si l'on cherche à adapter la preuve de Kupferman et Vardi [62], aux fonctions de coût, ce sont des automates compteur-faibles qui sont produits. Cela nous donne le théorème suivant.

Théorème 10.7 (Kuperberg, Vanden Boom [60])**.** *Les énoncés suivants sont effectivement équivalents pour une fonction de coût sur les arbres infinis :*

- *être accepté par un* $\mathtt{B} \wedge \mathtt{Büchi}$*-automate alternant compteur-faible,*
- *être à la fois accepté par un* $\mathtt{B} \wedge \mathtt{Büchi}$*-automate non-déterministe et par un* $\mathtt{S} \wedge \mathtt{Büchi}$*-automate non-détermniste.*

Le résultat complémentaire, et peut-être le plus surprenant, est le suivant.

Théorème 10.8 (Kuperberg, Vanden Boom [60])**.** *Sur les arbres infinis, les* $\mathtt{B} \wedge \mathtt{Büchi}$*-automates alternants compteur-faibles sont strictement plus expressifs que les* $\mathtt{B} \wedge \mathtt{Büchi}$*-automates alternants faibles.*

En effet, cet énoncé illustre une différence entre le cas des langages et des fonctions de coût. Dans ce dernier cas, deux classes distinctes sont à considérer, les fonctions de coût faibles et les fonctions de coût compteur-faible, alors que ces deux notions coïncident dans le cas des langages.

Une question naturelle s'ensuit :

> Existe-t-il une logique «naturelle» qui caractérise les fonctions de coût reconnues par les $\mathtt{B} \wedge \mathtt{Büchi}$-automates alternants compteur-faible ? Cette logique est-elle décidable, et plus précisément, est-elle plus facile à décider que la logique monadique de coût sur les arbres infinis ?

Nous donnerons des réponses à cette question au cours de la description de la piste 11.1.

2. Kuperberg et Vanden Boom ont introduit cette notion sous le nom d'automate **quasi-faible**. Nous adoptons ici la terminologie proposée par Jean-Éric Pin d'automate «compteur-faible», qui donne plus d'intuition sur la nature de la notion.

10.4 Effondrement sur les mots infinis

La logique monadique faible et la logique monadique sont d'expressivité différente sur les arbres infinis, mais, comme l'énonce le théorème 10.1 de McNaughton, ces deux logiques coïncident sur les mots infinis. Qu'en est-t-il de la logique monadique de coût faible et de la logique monadique de coût ?

Remarque 10.9 (Séparation sur les arbres). Soit L un langage d'arbres infinis tel que la fonction de coût χ_L est définissable en logique monadique de coût faible, disons par la formula φ. Cela signifie qu'il exists n tel que $L = \{u \ : \ [\![\varphi]\!](u) \leq n\}$. En utilisant la proposition 2.7 (ou plus exactement son adaptation directe à la logique monadique de coût faible), nous obtenons que L est définissable en logique monadique de coût faible.

Ainsi, supposons que L est un langage d'arbres infinis définissable en logique monadique, mais non définissable en logique monadique faible (par exemple le langage «il existe une branche le long de laquelle une infinité de a apparaissent»). Alors χ_L est définissable en logique monadique de coût (simplement en utilisant la même formule). Par contre, supposons que χ_L soit définissable en logique monadique de coût faible, alors, d'après la remarque précédente, L serait aussi accepté en logique monadique faible. Contradiction.

En revanche, la question est beaucoup moins évidente pour les mots infinis. En effet, il existe essentiellement deux méthodes pour montrer que la logique monadique peut être transformée en logique monadique faible sur les mots :

1. Il suffit de transformer la formule de la logique monadique en un automate de Muller déterministe (il s'agit de la construction de déterminisation de McNaugthon [70]). L'automate déterministe se traduit ensuite simplement en logique monadique faible.
2. Il suffit de remarquer qu'un langage de mots infinis reconnu par une formule de la logique monadique ainsi que son complémentaire sont tous deux acceptés par des automates de Büchi (conséquence des résultats fondateurs de Büchi [16]). Il s'ensuit, en utilisant le théorème 10.4 (qui lui est vrai dans le cas plus général des arbres infinis) que le langage est définissable en logique monadique faible.

Ces deux techniques sont inapplicables pour la logique monadique de coût, en effet :

1. Il n'existe pas de modèle déterministe d'automate (connu et utilisable) équivalent à la logique monadique de coût.
2. Toute fonction de coût sur les mots infinis définie par une formule de la logique monadique de coût est à la fois acceptée par un $\mathtt{B} \wedge \mathtt{Büchi}$-automate non-déterministe et par un $\mathtt{S} \wedge \mathtt{Büchi}$-automate non-déterministe (il s'agit de l'énoncé corresponant au cas des langages). En revanche, nous avons vu que la transposition du théorème 10.4, ne caractérise pas la classe des langages des fonctions de coût définissables en logique monadique faible, mais la classe strictement plus expressive des automates alternants compteur-faibles. La conclusion n'est pas celle escomptée.

Pour ces raisons, les techniques classiques sont inadaptées pour établir une relation d'équivalence entre la logique monadique de coût et sa variante faible sur les mots infinis. La preuve du théorème suivant nécessite pour ces raisons de nouvelles techniques.

Théorème 10.10 (Kuperberg,Vanden Boom [61])**.** *La logique monadique de coût faible et la logique monadique de coût ont effectivement la même expressivité sur les mots infinis.*

Ce dernier résultat clôt notre description des classes de fonctions régulières de coût faibles et compteur-faibles.

Quatrième partie

Conclusion

Chapitre 11

Pistes

Nous avons largement exposé au cours de ce document la notion de fonction régulière de coût et les résultats connus qui s'y rapportaient. Nous préférons, en guise de conclusion, présenter plusieurs pistes de recherche prolongeant ces travaux. En effet, de nombreuses questions restent encore en suspens, et il existe plusieurs directions pour des extensions pertinentes de ce modèle.

La première des pistes à explorer s'intéresse à la question ouverte principale sur les fonctions régulières de coût, à savoir la décidabilité de la domination entre fonctions calculées par des formules de la logique monadique de coût sur les arbres infinis (piste 11.1). Certains affaiblissements de cette questions sont considérés. Nous nous intéressons ensuite à la possibilité d'une formalisation autre des fonctions de coût au moyen de l'analyse non-standard (piste 11.2). La piste 11.3 s'intéresse à la résolution de problèmes dans lesquels plusieurs bornes sont impliquées simultanément. L'application des fonctions régulières de coût à la vérification de modèles est envisagée dans le piste 11.4. La résolution de problèmes de décision plus fins que la domination est ensuite considéré (piste 11.5). Nous omettons ici d'autre questions, en particulier traitant de la relation entre la logique monadique de coût et la logique MSO + $\mathbb{U}$.

11.1 Vers la résolution des fonctions de coût sur les arbres infinis

Comme nous l'avons vu au cours de ce document, le grand problème ouvert de la théorie des fonctions régulières de coût est la décidabilité de la domination sur les arbres infinis. Ce problème s'avère complexe. Il existe pourtant plusieurs types de résultat, en un sens, plus faibles, qui méritent d'être étudiés, et qui sont susceptibles d'apporter quelques idées quant à la résolution du cas général.

Nous avons vu le résultat caractérisant la logique monadique de coût faible sur les arbres infinis (théorème 10.5), et le résultat de décidabilité qui lui est lié. Nous avons vu

également la notion d'automates à coût compteur-faibles. La motivation était de caractériser les fonctions de coût simultanément $\mathtt{B} \wedge \mathtt{Büchi}$ et $\mathtt{S} \wedge \mathtt{Büchi}$ (theorem 10.7). Une question naturelle est de déterminer si une logique (naturelle) correspond à la notion d'automates compteur-faibles.

La réponse à cette question est positive. Pour l'énoncer, il convient de définir la notion de point-fixe borné. Considérons, une formule $\varphi(X, y)$ possédant, entre autres, une variable libre X monadique, et une variable libre du premier ordre y. Nous supposons que les prédicats de la forme $t \in X$ apparaissent positivement dans la formule. Le dépliage de cette formule à l'ordre n est défini par récurrence par :

$$\varphi^0_{X,y}(z) = \bot \qquad \text{et} \quad \varphi^{n+1}_{X,y}(z) = \varphi[y \leftarrow z][t \in X \leftarrow \varphi^n_{X,y}(t)] \ ,$$

où la notation $\varphi[t \in X \leftarrow \varphi^n(t)]$ signifie que l'on substitue dans φ à chaque occurrence de la construction $t \in X$ (pour un t quelconque) la formule $\varphi^n(t)$. Remarquons que l'hypothèse de positivité dans l'apparition des tests $t \in X$ a deux conséquences. La première est que $\varphi^n \Rightarrow \varphi^{n+1}(t)$. La seconde est que si φ est une formule monadique de coût, il en va de même pour $\varphi^n(t)$, en particulier en ce qui concerne la positivité des prédicats de cardinal.

Ajoutons maintenant l'opérateur de **point-fixe borné** à la logique monadique de coût (faible), *c.-à-d.* qu'il devient possible d'utiliser la construction $\varphi^N_{X,y}(z)$ dans la logique, pourvu que cette construction apparaisse positivement. Comme pour la logique monadique de coût, la sémantique d'une formule φ de cette logique est définie comme :

$$[\![\varphi]\!](\mathcal{S}) = \inf\{n \ : \ \mathcal{S}, N = n \models \varphi\} \ .$$

Ainsi, cette logique permet non seulement de vérifier que le cardinal de certains ensembles ne dépasse pas n, mais également permet de garantir que certains éléments apparaissent rapidement dans un point-fixe, *c.-à-d.* après moins de n itérations du point-fixe.

Exemple 11.1. Par exemple, la formule

$$\forall x_1 \forall x_2 \ \ [y = x_1 \vee \exists z \in X \ \mathtt{arc}(z, y)]^N_{X,y}(x_2) \ ,$$

exprime le diamètre d'un graphe. Ainsi, la fonction de coût est la même que dans l'exemple 2.5, mais est encodée de manière très différente.

Remarque 11.2. Une construction similaire permet, si la structure possède un ordre total définissable (il s'agit en particulier du cas des mots et des arbres, finis ou infinis), de définir le prédicat $|X| \leq N$ au moyen d'un point-fixe borné. Ainsi, sur de telles structures, il n'est pas nécessaire d'utiliser de prédicats de cardinal.

Sur les mots finis ou infinis et sur les arbres finis (il s'agit d'une conséquence du théorème suivant dans le cas pathologique des structures finies), cette logique a la même expressivité que la logique monadique de coût. En revanche, sur les structures finies en général, la question est ouverte, et la réponse est vraisemblablement négative : il existe des formules de la logique monadique de coût avec point-fixe borné pour lesquelles il n'existe pas de formule de la logique monadique de coût équivalente.

Théorème 11.3 (Blumensath, C., Kuperberg, Vanden Boom, non publié)**.** *Les propriétés suivantes sont équivalentes pour une fonction de coût sur les arbres infinis :*

- *être définissable par un* $\mathtt{B} \wedge \mathtt{Büchi}$*-automate alternant compteur-faible,*
- *être définissable en logique monadique de coût faible avec point-fixe borné.*

De plus, la domination pour les fonctions définissables en logique monadique de coût faible avec point-fixe borné est décidable.

Ce résultat nous donne donc accès à une logique plus expressive que la logique monadique de coût faible, tout en gardant la décidabilité sur les arbres infinis.

Ensembles fins. Dans l'objectif d'établir la décidabilité de la logique monadique de coût sur les arbres infinis, un autre cas intéressant mérite d'être étudié, celui des arbres fins. Un ensemble de nœuds d'un arbre est dit **fin** («thin») si, quand il est clos vers le haut, il ne contient qu'un nombre dénombrable de branches. Un arbre est **fin** si l'ensemble de ses nœuds est fin. En particulier, un arbre est fin si et seulement s'il ne contient pas d'arbre binaire complet induit.

Les arbres fins et les ensembles fins sont significativement plus simples que les ensembles de nœuds généraux d'un arbre. Pour cette raison, il est probable que les deux problèmes suivant soient plus faciles à établir que la décidabilité de la logique monadique de coût sur les arbres infinis généraux.

Question ouverte 11.4. *Le problème de la domination entre formules monadiques de coût sur les arbres infinis fins est-il décidable ?*

La seconde question est plus générale.

Question ouverte 11.5. *Le problème de la domination entre formules monadiques de coût dans lesquelles la quantification ne porte que sur des ensembles fins est-il décidable sur les arbres infinis ?*

11.2 Analyse non-standard et théorie interne des ensembles

Beaucoup de raisonnements dans la théories des fonctions de coût sont réalisés, au niveau intuitif, au moyen de descriptions comme «le nombre d'occurrences de la lettre a est grand» ou «l'intervalle délimité par des b consécutifs est petit.» En pratique, c'est toujours à ce niveau que les preuves sont cherchées. L'apparition de fonctions de corrections reflète la matérialisation en langage mathématique de ces concepts.

Une question naturelle est donc : est-t-il possible de faire mieux, et de rester dans les preuves plus proches de cette intuition ? Ainsi, est-il possible de raisonner formellement en utilisant des concepts comme «grand» et «petit» ?

L'approche profinie de Toruńczyk offre une première réponse à cette question : il est possible de voir les fonctions régulières de coût comme des langages de mots profinis (voir la section 1.5). Dans cette terminologie beaucoup de notions sont simplifiées. Cette idée est

à l'origine de l'approche «non-standard» que nous développons maintenant, et qui offre une alternative à l'utilisation des langages de mots profinis.

Des questions similaires existent depuis l'origine de l'analyse, quand Leibniz, puis Euler et Cauchy faisaient usage de quantités «infiniment petites» dans leurs raisonnements, aucun fondement rigoureux n'était alors donné pour justifier l'utilisation de ces objets. Le développement rigoureux de l'analyse menée, entres autres, par Dedekind et Weierstrass, a progressivement cantonné la notion d'infiniment petit au discours informel. La notation privilégiée devint alors d'utiliser des variables ϵ, η pour faire tendre explicitement certaines quantités vers zéro. L'analyse non-standard, initiée par Robinson en 1961 [90], propose un cadre formel rigoureux dans lequel la notion d'infiniment petit prend un sens précis. Cela prend en particulier la forme d'un corps algébriquement clos, les **hyperréels** ${}^*\mathbb{R}$, qui étendent strictement les réels, et contient de nouveaux éléments interprétés comme des «infiniment petits» et des «infiniment grands». Dans ce cadre, les notions comme la continuité s'expriment par des énoncés similaires à [1] «pour tous hyperréels $x \simeq y$, $f(x) \simeq f(y)$» ou $x \simeq y$ signifie que $x-y$ est un infiniment petit. Beaucoup de raisonnements sont grandement simplifiés quand ils sont réalisés dans les hyperréels plutôt que dans les réels, et l'analyse moderne fait un usage intensif de cette approche.

En 1977, Nelson propose une description axiomatique de l'analyse non-standard, la **théorie interne des ensembles** («internal set theory», ou simplement **IST**) [80]. L'IST est une extension de la théorie des ensembles et de ZFC en particulier. Dans ZFC, tous les objets sont des ensembles, et une seule relation permet de les comparer, l'appartenance «$\in$». Les axiomes de la théorie énoncent les propriétés de la relation $\in$. Dans IST, une nouvelle notion peut être exprimée : certains objets x (ensembles) sont «standards», ce qui est noté simplement $\mathtt{St}(x)$, quand d'autres ne le sont pas. Intuitivement, les objets standards sont les objets accessibles, alors que les objets non-standards en sont un prolongement inaccessible. Tous les axiomes de ZFC restent valides dans IST. De nouveaux axiomes, permettant de raisonner avec la notion d'être «standard», sont également disponibles. Travailler dans IST plutôt que dans ZFC n'apporte rien à ZFC en terme de pouvoir de preuve (on dit qu'il s'agit d'une extension **conservative**). En effet, tout énoncé «classique», *c.-à-d.* ne faisant pas référence à la nouvelle notion «être standard», peut être prouvé dans ZFC si et seulement si il peut être prouvé dans IST. L'apport d'IST est que l'utilisation du nouveau prédicat $\mathtt{St}(x)$ permet de simplifier énormément certaines preuves d'énoncés classiques.

Dans IST, toute notion classique est autant valable que dans ZFC, et ne peut définir (c'est une conséquence des axiomes) que des objets standards. Ainsi, $\mathbb{N}, \mathbb{R}, 1, 2, \sqrt{2}, \pi, \ldots$ sont des objets standards. Par contre, bien que $\mathbb{N}$ soit un ensemble standard, tous les enters ne sont pas standards. De nouveaux entiers apparaissent qu'il faut comprendre comme «arbitrairement/infiniment grands.» De même, certains réels sont non-standards. Ceux-

1. Cette description est incorrecte en tant que telle. En particulier, f est une fonction des réels dans les réels, et il faut donc d'abord l'étendre en une fonction des hyperréels dans les hyperréels ${}^*f$. De plus, cet énoncé exprime la continuité uniforme et non simplement la continuité. Pour exprimer la continuité, il suffit de restreindre x à prendre ses valeurs parmi les réels (et non les hyperréels).

ci peuvent être vus comme des infiniment petits (plus généralement des réels infiniment proches de réels standards), et des infiniment grands.

La force de l'IST est que cette extension est faite au niveau de l'axiomatique, et toutes les objets mathématiques sont accessibles, certains pouvant maintenant prendre une forme non-standard. Ainsi, il existe naturellement des mots finis standards, et des mots finis non-standards, il existe des arbres infinis standards, et des arbres infinis non-standards, etc... Bien qu'il soit hors de propos ici de développer ces notions plus en détails, nous allons essayer de mettre à profit ces nouvelles possibilités pour décrire les fonctions de coût.

Pour commencer, plaçons-nous dans un contexte général. Considérons une formule φ (du langage mathématique, pas d'une logique en particulier) **classique** (*c.-à-d.* qui ne fait pas usage du mot clef «standard»), et qui est **monotone** en une variable entière n, (*c.-à-d.* $\varphi(n) \Rightarrow \varphi(n+1)$ pour tout n, ce qui peut être garanti de manière syntaxique en autorisant l'usage de la variable n uniquement dans des prédicats de la forme «$? \leq n$» apparaissant positivement dans la formule). Nous supposons dans la suite que φ est toujours une telle formule. Nous noterons $\exists^{\mathtt{St}} n\ \varphi$ le fait qu'il existe un entier standard n tel que φ est satisfaite, et $\forall^{\neg\mathtt{St}} n\ \varphi$ le fait que pour tout n non-standard, φ est satisfaite.

Alors, une conséquence simple des axiomes d'IST est :

Proposition 11.6. *Si φ n'a que des paramètres standards, alors*

$$\exists n\ \varphi \quad \Leftrightarrow \quad \exists^{\mathtt{St}} n\ \varphi\ .$$

De plus, même si certains paramètres de φ prennent des valeurs non-standards,

$$\exists^{\mathtt{St}} n\ \varphi \quad \Leftrightarrow \quad \forall^{\neg\mathtt{St}} n\ \varphi\ .$$

Nous allons appliquer ces faits pour montrer comment simplifier les questions d'existence de borne. Rien ne nous oblige de se placer dans le cadre restreint de la logique monadique de coût. Ainsi, nous nous intéressons à décider (en fait, simplement reformuler) la véracité d'une formule

$$\exists n\ \bar{Q}\bar{x}\ \varphi,$$

où $\bar{Q}\bar{x}$ est une séquence de variable quantifiée, et φ est une combinaison booléenne positive de prédicats $R(\bar{y})$ n'impliquant que des variables $\bar{y}$ de $\bar{x}$, et de prédicats $f(\bar{y}) \leq n$, où f est une fonction dans les entiers et les variables $\bar{y}$ appartiennent à $\bar{x}$. Les prédicats $R, \ldots,$ et $f, \ldots$ doivent être standards.

Le problème de l'existence de borne pour une fonction $\bar{Q}\bar{X}\ \psi$ de la logique monadique de coût sur les mots (bien que cela ne change rien) est une formule de cette forme. En effet, il s'agit de résoudre (en simplifiant les notations) :

$$\exists n\ \forall u \in \mathbb{A}^*\ \bar{Q}\bar{X}\ \psi\ .$$

Ainsi, sans ce cas, la seule fonction f nécessaire est la fonction cardinal $X \mapsto |X|$.

Il se trouve que, en utilisant la première équivalence énoncée dans la proposition ci-dessus, il est équivalent à $\exists n\ \bar{Q}\bar{x}\ \varphi$ de résoudre :

$$\exists^{\mathtt{St}} n\ \bar{Q}\bar{x}\ \varphi.$$

Or, la deuxième partie de la proposition permet d'exprimer l'existence d'un n standard par une quantification universelle. Il s'ensuit que $\exists^{\mathtt{St}} n$ commute avec les quantifications existentielles et universelles. Ainsi, il est équivalent de résoudre :

$$\bar{Q}\bar{x}\ \exists^{\mathtt{St}} n\ \varphi\ .$$

Enfin, une dernière analyse nous montre qu'il est équivalent de résoudre la formule

$$\bar{Q}\bar{x}\ \varphi^{\mathtt{St}}\ ,$$

où $\varphi^{\mathtt{St}}$ est la formule φ dans laquelle chaque prédicat $f(\bar{y}) \leq n$ est remplacée par $\mathtt{St}(f(\bar{y}))$.

Ainsi, en utilisant l'approche non-standard, chaque propriété quantitative $|X| \leq n$, qui est de nature «quantitative», est naturellement remplacée par un prédicat $\mathtt{St}(|X|)$ qui lui est un prédicat booléen, non-standard. Vue sous cet angle, la logique monadique de coût n'est rien d'autre que la logique monadique usuelle, dans laquelle il est possible d'exprimer qu'un ensemble est de taille standard, ce test n'étant autorisé à apparaitre que positivement.

Bien entendu, nous n'avons pour l'instant que reformulé les questions étudiées par les fonctions régulières de coût dans le cadre de la théorie interne des ensembles. Le vrai travail reste à faire, à savoir, mener les preuves dans ce nouveau cadre. Il est probable que de nombreux points techniques peuvent être simplifiés de la sorte.

11.3 Une extension des fonctions de coût à plusieurs ordres de grandeur

Nous avons vu, au cours des développements de ce document, qu'il était utile, parfois, de considérer des problèmes impliquant plusieurs bornes simultanément. C'est en particulier le cas quand il s'agit de décider de la domination d'une fonction f par une fonction g, disons sur l'ensemble des mots $\mathbb{A}^*$:

$$f \preccurlyeq g \qquad \text{si et seulement si} \qquad \forall m \exists n\ \forall u \in \mathbb{A}^*\ g(u) \leq m \to f(u) \leq n\ .$$

Imaginons que φ et ψ soient des formules de la logique monadique de coût définissant respectivement $f = [\![\varphi]\!]$ et $g = [\![\psi]\!]$, et dénotons par $\varphi[m]$, la formule φ dans laquelle la constante N a été remplacée par m (et de manière similaire pour $\psi[n]$). On a alors $u \models \varphi[n]$ si et seulement si $[\![f]\!](u) \leq m$ (nous nous autorisons ici quelques licenses de notation qui ne devraient pas nuire à la compréhension).

La propriété de domination peut alors se récrire comme

$$\forall m \exists n \ \forall u \in \mathbb{A}^* \ (u \models \psi[m]) \to (u \models \varphi[n]) \ .$$

ou encore,

$$\forall m \exists n \ \forall u \in \mathbb{A}^* \ (u \models \xi) \qquad \text{où} \quad \xi \stackrel{\texttt{def}}{=} \neg\psi[m] \vee \varphi[n] \ .$$

La formule ξ est une formule dont la syntaxe est celle de la logique monadique, augmentée de la possibilité d'utiliser le prédicat $|X| \leq n$ and il apparait positivement dans la formule, et le prédicat $|X| \leq m$ quand il apparait négativement dans la formule. L'extension de la logique monadique de coût à plusieurs ordres de grandeurs généralise ce type d'énoncés à des formules quelconques, et à des contextes contenant un nombre arbitraire de variables.

Ainsi, fixons nous une série de quantifications $\forall n_1 \exists n_2 \forall n_3 \ldots$, que nous abrègerons simplement en $\bar{Q}\bar{n}$. Une formule de la **logique monadique de coût à plusieurs ordres de grandeur** dans le contexte de quantification $\bar{Q}\bar{n}$, ou plus simplement une $\bar{Q}\bar{n}$**-formule**, est une formule de la logique monadique augmentée par la possibilité d'utiliser les prédicats de la forme $|X| \leq n_i$ apparaissant $(\exists)$ positivement si la variable n_i est quantifiée existentiellement dans $\bar{Q}\bar{n}$, et $(\forall)$ négativement si la variable n_i est quantifiée universellement dans $\bar{Q}\bar{n}$. Ainsi, les formules de la logique monadique sont des ε-formules, les formules de la logique monadique de coût sont des $\exists N$-formules, et leurs complémentaires sont des $\forall N$-formules. La formule ξ ci-dessus est une $\forall m \exists n$-formule.

Il est naturel d'élever la totalité de la théorie des fonctions de coût régulières à ce cadre plus général. Par exemple le résultat de décidabilité sur les mots est le suivant :

Théorème 11.7 (non publié)**.** *Le problème, étant donné une $\bar{Q}$-formule φ, de décider si «$\models \bar{Q}\bar{n} \ \forall u \in \mathbb{A}^* \ \varphi$» est décidable.*

En appliquant ce résultat à la formule ξ ci-dessus, cela nous redonne une preuve de la domination pour les formules de la logique monadique de coût.

Pourquoi appeler cette logique «à plusieurs ordres de grandeur ?» C'est assez naturel. Une formule $\bar{Q}\bar{n} \ \Psi$ (ici, le fait que Ψ soit d'une logique particulière n'a pas réellement d'importance) peut être vue comme un jeu. Le jeu consomme la formule depuis l'extérieur. Quand une variable est quantifiée existentiellement, le joueur existentiel joue en choisissant la valeur de la variable, et quand une quantification universelle est rencontrée, le joueur universel choisit la valeur de la variable. Quand toutes les quantifications ont été consommées, si la formule Ψ est satisfaite pour cette valuation des variables de $\bar{n}$, le joueur existentiel gagne, sinon le joueur universel est vainqueur. Il est intuitif et classique que ce jeu modélise l'évaluation de la formule.

Si maintenant Ψ utilise de manière positive les variables quantifiées existentiellement, et de manière négative les variables quantifiées universellement, cela transparait dans les stratégies possibles des joueurs. En effet, cela signifie que, quand un joueur choisit la valeur d'une variable n_i au cours du jeu, il a tout intérêt à la choisir la plus grande possible. Ainsi,

si la quantification est de la forme $\exists n_1 \forall n_2 \exists n_3$, alors le joueur existentiel a tout intérêt à choisir n_1 très grand. À cela, le joueur universel a tout intérêt à répondre par n_2 très grand, et en particulier très grand devant n_1. Enfin, le joueur existentiel choisissant n_3 a encore intérêt, lui aussi, à choisir n_3 très grand, cette fois-ci devant n_1 et n_2. En ce sens, il y a l'ordre de grandeur «des constantes et de tout ce qu'elle peuvent produire», et il faut considérer que n_1 est au delà de cette plage de valeur. Ainsi, n_1 est d'un ordre de grandeur supérieur à toutes les «constantes». En suivant la même intuition n_2 est d'un ordre de grandeur supérieur à n_1, et n_3 est d'un ordre de grandeur supérieur à n_2 (et par conséquent aussi n_1). Cette approche par ordre de grandeur généralise l'approche intuitive «petit/grand» utilisée pour les fonctions régulières de coût.

Cette vision à plusieurs ordres de grandeur se marie très bien avec la théorie interne des ensembles que nous avons évoquée comme piste précédente. Plus précisément, il existe des extensions de l'IST qui autorisent de manipuler plusieurs ordres de grandeur. Il s'agit en particulier de la **théorie interne des ensembles relative** («relative internal set theory», **RIST**) [84] (dont il existe plusieurs variantes). Dans cette extension, au lieu d'un unique prédicat unaire `St` permettant de séparer les objets standards des objets non-standards, une relation binaire $\sqsubseteq$ est utilisée. L'expression $x \sqsubseteq y$ signifie «x est plus standard que y». La possibilité de manipuler des objets standards à différent niveaux est exactement ce qu'il manque pour pouvoir traiter des fonctions de coût à plusieurs ordres de grandeur. Le gain apporté par l'utilisation de l'analyse non-standard est vraisemblablement encore plus sensible pour ces questions impliquant plusieurs ordres de grandeur.

11.4 Jeux et vérification de modèles

L'une des conséquences de la théorie des langages réguliers sur les modèles infinis est qu'elle propose des outils permettant d'aborder un certain nombre de questions en vérification de manière uniforme. Cette approche générique est souvent optimale. Citons en particulier les problèmes de **vérification de modèle** («model-checking»). Par exemple, il est possible de définir un **jeu à pile** comme un jeu infini, dont les positions sont les configurations d'un automate à pile. Les coups possibles sont les transitions de l'automate. Un ensemble régulier de configurations appartient au jouer Ève et le reste appartient à Adam. Un tel jeu à pile peut être équipé d'une condition de gain, par exemple de parité en donnant l'ensemble régulier des configurations de priorité 1, l'ensemble régulier des configurations de priorité 2, etc...

Théorème 11.8 (conséquence de [76], et [105] pour la complexité)**.** *Décider du vainqueur dans un jeu à pile de parité est DEXPTIME.*

Ce thème de recherche a connu un regain d'intérêt quand des conditions de gain non-régulières ont été considérées, et plus précisément des conditions comme «au cours de la partie, la taille de la pile n'est pas bornée» , ainsi que les combinaisons de cette condition avec les conditions de parité [19, 14, 39, 94].

L'utilisation des fonction régulières de coût offre de nombreuses perspectives de continuations de ces questions. Citons en particulier :

- Peut-on résoudre un jeu à pile dont l'objectif est une fonction régulière de coût sur les parties. Ainsi, l'objectif d'Ève est de produire un entier n et une stratégie σ_{E} dans le jeu garantissant que pour toute partie compatible avec cette stratégie, l'objectif ne dépasse pas la valeur n.
- Dans les extensions mentionées ci-dessus, la taille de la pile peut être remplacée par une fonction (régulière de coût) de la pile.

D'autres variations prennent leur sens dans le cadre quantitatif.

11.5 Vers une analyse plus fine

Beaucoup de travaux en analyse mathématique peuvent être compris comme établissant des inégalités entre différentes quantités. Ces résultats peuvent être ordonnés suivant leur finesse. Tao présente par exemple une telle classification en (1) inégalités exactes, (2) inégalités presque exactes, (3) quasi-inégalités, (4) quasi-inégalités avec pertes logarithmiques, (5) quasi-inégalités avec pertes polynomiales, et (6) inégalités grossières [99]. Les inégalités exactes sont de la forme $X \leq Y$, ou $X \leq CY$, où C est une constante comme e ou π. Les inégalités grossières sont de la forme $X \leq f(Y)$ pour f une fonction non contrôlée. Cette description offre une gradation, du plus précis au plus grossier. Il est courant, et même nécessaire dans le développement de résultats, d'établir d'abord les résultats dans leur forme grossière, puis de, petit à petit, en améliorer la précision. Bien entendu, les preuves deviennent alors de plus en plus difficiles et techniques.

Une description similaire s'applique aux fonctions régulières de coût. Ainsi, nous pouvons paraphraser la taxonomie ci-dessus pour classifier les problèmes de décision impliquant la comparaison de fonctions, ici, de deux fonctions f, g d'un ensemble $\mathbb{A}^*$ dans $\mathbb{N} \cup \{\infty\}$.

1. Les **problèmes d'inégalités exactes** sont de la forme «$f \leq g$?», ou encore «$f \leq Cg$?» pour une constante C fixée. Ces problèmes sont indécidables pour les fonctions calculées par des automates de distance (théorème 1.13 de Krob).

2. Les **problèmes d'inégalités avec perte affine** sont de la forme «existe-t-il C tel que $f \leq Cg$?». Ce problème est ouvert dans le cas de fonctions calculées par des B-automates ou des formules monadiques de coût. Il est décidable si f et g sont représentées par des automates de distance. Le résultat est plus précisément que, si $f \preccurlyeq g$, alors il existe C tel que $f \leq Cg$ [29]. Comme $f \leq Cg$ implique $f \preccurlyeq g$ et que $f \preccurlyeq g$ est décidable, il s'ensuit qu'il suffit de décider $f \preccurlyeq g$ pour décider le problème d'inégalité avec perte affine correspondant.

3. Les **problèmes d'inégalités avec perte polynomiale** sont de la forme «existe-t-il un polynôme α tel que $f \leq \alpha \circ g$?». Ce résultat est décidable si f et g sont représentées par des formules monadiques de coût ou des automates à coût. En effet, nous avons vu que si $f \preccurlyeq g$, alors $f \preccurlyeq_\alpha g$ pour α polynomial. Le problème reste ouvert dans le cas des arbres.

En effet, une «perte» exponentielle est produite lors du codage des formules de la logique monadique de coût en automates.

4. Les **problèmes d'inégalités grossières** sont de la forme $f \preccurlyeq g$. Leur décidabilité est le sujet même de ce document.

Cette première classification des résultats de décidabilité présente son intérêt, mais une réponse plus pragmatique est possible. En effet, la «vraie question» que l'on se pose n'est pas de disposer d'un algorithme qui décide une autre relation que l'inégalité $f \leq g$, mais bien de disposer d'un algorithme qui approxime la comparaison exacte. Cette approche nous aiguille vers une autre classification.

1. La **comparaison exacte**, «$f \leq g$?», est celle que nous avons vue précédemment, et qui est indécidable.

2. La **comparaison quasi-exacte**, étant donné f, g et $\varepsilon > 0$, consiste à produire un algorithme qui,
– s'il répond oui, garantit que $f \leq (1+\varepsilon)g$,
– s'il répond non, garantit que $f \not\leq g$.
Il s'agit du point de départ de la thèse de Laure Daviaud. Un premier résultat donne une réponse positive à cette question dans le cas des automates de distance.

Théorème 11.9. *([29]) Le problème de la comparaison quasi-exacte entre fonctions calculées par automates de distance est décidable.*

L'algorithme proposé est EXPSPACE, mais la conjecture est que ce résultat est PSPACE. Le problème est ouvert pour d'autres descriptions des fonctions, comme les automates à coût ou les fonctions définies par les formules de la logique monadique de coût.

3. La **comparaison avec approximation affine** consiste à produire un algorithme tel que :
– s'il répond oui, alors $f \leq Cg$ pour un certain C,
– s'il répond non, alors $f \not\leq g$.
Bien entendu, tout algorithme qui décide du problèmes d'inégalités avec perte affine satisfait cette spécification. Une telle comparaison est donc possible pour les automates de distance. Les autres cas sont ouverts.

4. La **comparaison avec approximation polynomiale** consiste à produire un algorithme tel que :
– s'il répond oui, alors $f \leq \alpha \circ g$ pour un polynôme α,
– s'il répond non, alors $f \not\leq g$.
Là encore il s'agit d'un problème plus facile que l'inégalité avec perte polynomiale. Les résultats de décidabilité sont donc hérités de l'inégalité avec perte polynomiale.

5. La **comparaison avec approximation grossière** est identique, mais ne requiert pas que la fonction α soit polynomiale.

Beaucoup de ces questions sont ouvertes. Elles le sont encore plus si les questions de complexité sont abordées. Il s'agit donc d'une prolongation importante et naturelle aux fonctions régulières de coût. Les techniques doivent être grandement affinées pour les aborder.

Bibliographie

[1] Sebastian Bala. Regular language matching and other decidable cases of the satisfiability problem for constraints between regular open terms. In Volker Diekert and Michel Habib, editors, *STACS 2004 : 21st International Symposium on Theoretical Aspects of Computer Science*, volume 2996 of *Lecture Notes in Comput. Sci.*, pages 596–607. Springer, 2004.

[2] Achim Blumensath, Martin Otto, and Mark Weyer. Boundedness of monadic second-order formulae over finite words. In Susanne Albers, Alberto Marchetti-Spaccamela, Yossi Matias, Sotiris E. Nikoletseas, and Wolfgang Thomas, editors, *ICALP 2009 (2) : Automata, Languages and Programming, 36th International Colloquium*, volume 5556 of *Lecture Notes in Comput. Sci.*, pages 67–78. Springer, 2009.

[3] Achim Blumensath, Martin Otto, and Mark Weyer. Decidability results for the boundedness problem. Available online, 2012.

[4] Mikołaj Bojańczyk. A bounding quantifier. In Jerzy Marcinkowski and Andrzej Tarlecki, editors, *CSL 2004 : Computer Science Logic, 18th International Workshop*, volume 3210 of *Lecture Notes in Comput. Sci.*, pages 41–55. Springer, 2004.

[5] Mikołaj Bojańczyk. Factorization forests. In Volker Diekert and Dirk Nowotka, editors, *Developments in Language Theory*, volume 5583 of *Lecture Notes in Comput. Sci.*, pages 1–17. Springer, 2009.

[6] Mikołaj Bojańczyk. Weak MSO with the unbounding quantifier. In Susanne Albers and Jean-Yves Marion, editors, *STACS 2009 : 26th International Symposium on Theoretical Aspects of Computer Science*, volume 3 of *LIPIcs*, pages 159–170. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, Germany, 2009.

[7] Mikołaj Bojańczyk and Thomas Colcombet. Bounds in ω-regularity. In *LICS 06 : 21th IEEE Symposium on Logic in Computer Science*, pages 285–296. IEEE Computer Society, 2006.

[8] Mikołaj Bojańczyk and Thomas Colcombet. Bounds in ω-regularity. *Log. Methods Comput. Sci.*, page 54, 2009.

[9] Mikołaj Bojańczyk and Szymon Toruńczyk. Deterministic automata and extensions of weak MSO. In Ravi Kannan and K. Narayan Kumar, editors, *FSTTCS 2009 :*

IARCS Annual Conference on Foundations of Software Technology and Theoretical Computer Science, volume 4 of *LIPIcs*, pages 73–84. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2009.

[10] Mikołaj Bojańczyk and Szymon Toruńczyk. Weak MSO+U over infinite trees. In Christoph Dürr and Thomas Wilke, editors, *STACS 2012 : 29th International Symposium on Theoretical Aspects of Computer Science*, volume 14 of *LIPIcs*, pages 648–660. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2012.

[11] Udi Boker. Good for games Büchi automata and memorylessness. Communication personnelle, 2009.

[12] Michael Vanden Boom. Weak cost monadic logic over infinite trees. In Filip Murlak and Piotr Sankowski, editors, *MFCS 2011 : Mathematical Foundations of Computer Science*, volume 6907 of *Lecture Notes in Comput. Sci.*, pages 580–591. Springer, 2011.

[13] Michael Vanden Boom and Denis Kuperberg. On the expressive power of cost logics over infinite words. In Artur Czumaj, Kurt Mehlhorn, Andrew M. Pitts, and Roger Wattenhofer, editors, *ICALP 2012 (2) : Automata, Languages, and Programming - 39th International Colloquium*, volume 7392 of *Lecture Notes in Comput. Sci.*, pages 287–298. Springer, 2012.

[14] Alexis-Julien Bouquet, Olivier Serre, and Igor Walukiewicz. Pushdown games with unboundedness and regular conditions. In Paritosh K. Pandya and Jaikumar Radhakrishnan, editors, *FSTTCS 2003 : Foundations of Software Technology and Theoretical Computer Science, 23rd Conference*, volume 2914 of *Lecture Notes in Comput. Sci.*, pages 88–99. Springer, 2003.

[15] J. Richard Büchi. Weak second-order arithmetic and finite automata. *Z. Math. Logik Grundlagen Math.*, 6 :66–92, 1960.

[16] J. Richard Büchi. On a decision method in restricted second order arithmetic. In *Logic, Methodology and Philosophy of Science (Proc. 1960 Internat. Congr .)*, pages 1–11. Stanford Univ. Press, Stanford, Calif., 1962.

[17] J. Richard Büchi and Lawrence H. Landweber. Solving sequential conditions by finite-state strategies. *Trans. Amer. Math. Soc.*, 138 :295–311, 1969.

[18] Jérémie Cabessa, Jacques Duparc, Alessandro Facchini, and Filip Murlak. The Wadge hierarchy of max-regular languages. In Ravi Kannan and K. Narayan Kumar, editors, *FSTTCS 2009 : IARCS Annual Conference on Foundations of Software Technology and Theoretical Computer Science*, volume 4 of *LIPIcs*, pages 121–132. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2009.

[19] Thierry Cachat, Jacques Duparc, and Wolfgang Thomas. Solving pushdown games with a sigma3 winning condition. In Julian C. Bradfield, editor, *CSL 2002 : Computer Science Logic, 16th International Workshop*, volume 2471 of *Lecture Notes in Comput. Sci.*, pages 322–336. Springer, 2002.

[20] Arnaud Carayol, Christof Löding, Damian Niwiński, and Igor Walukiewicz. Choice functions and well-orderings over the infinite binary tree. *Central European Journal of Mathematics*, 8(4) :662–682, 2010.

[21] Olivier Carton, Thomas Colcombet, and Gabriele Puppis. Regular languages of words over countable linear orderings. In Luca Aceto, Monika Henzinger, and Jiri Sgall, editors, *ICALP 2011 (2) : Automata, Languages and Programming - 38th International Colloquium*, volume 6756 of *Lecture Notes in Comput. Sci.*, pages 125–136. Springer, 2011.

[22] Olivier Carton and Ramón Maceiras. Computing the Rabin index of a parity automaton. *Theoret. Comput. Sci.*, 33(6) :495–506, 1999.

[23] Olivier Carton, Dominique Perrin, and Jean-Éric Pin. Automata and semigroups recognizing infinite words. In Jörg Flum, Erich Grädel, and Thomas Wilke, editors, *Logic and Automata : History and Perspectives [in Honor of Wolfgang Thomas]*, volume 2 of *Texts in Logic and Games*, pages 133–168. Amsterdam University Press, 2008.

[24] Thomas Colcombet. Factorisation forests for infinite words. In Erzsébet Csuhaj-Varjú and Zoltán Ésik, editors, *FCT 2007 : Fundamentals of Computation Theory, 16th International Symposium*, volume 4639 of *Lecture Notes in Comput. Sci.*, pages 226–237. Springer, 2007.

[25] Thomas Colcombet. The theory of stabilisation monoids and regular cost functions. In Susanne Albers, Alberto Marchetti-Spaccamela, Yossi Matias, Sotiris E. Nikoletseas, and Wolfgang Thomas, editors, *ICALP 2009 (2) : Automata, Languages and Programming, 36th International Colloquium*, volume 5556 of *Lecture Notes in Comput. Sci.*, pages 139–150. Springer, 2009.

[26] Thomas Colcombet. Forms of determinism for automata. In Christoph Dürr and Thomas Wilke, editors, *STACS 2012 : 29th International Symposium on Theoretical Aspects of Computer Science*, volume 14 of *LIPIcs*, pages 1–23. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2012.

[27] Thomas Colcombet. The factorisation forest theorem. To appear in the handbook "Automata : from Mathematics to Applications", 2013.

[28] Thomas Colcombet. Regular cost functions, part I : logic and algebra over words. *Log. Methods Comput. Sci.*, page 47, 2013.

[29] Thomas Colcombet and Laure Daviaud. Approximate comparison of distance automata. In Natacha Portier and Thomas Wilke, editors, *STACS 2013 : 30th International Symposium on Theoretical Aspects of Computer Science*, volume 20 of *LIPIcs*, pages 574–585. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2013.

[30] Thomas Colcombet, Denis Kuperberg, and Sylvain Lombardy. Regular temporal cost functions. In Samson Abramsky, Cyril Gavoille, Claude Kirchner, Friedhelm Meyer

auf der Heide, and Paul G. Spirakis, editors, *ICALP 2010 (2) : Automata, Languages and Programming, 37th International Colloquium*, volume 6199 of *Lecture Notes in Comput. Sci.*, pages 563–574. Springer, 2010.

[31] Thomas Colcombet and Christof Löding. The nesting-depth of disjunctive μ-calculus for tree languages and the limitedness problem. In Michael Kaminski and Simone Martini, editors, *CSL 2008 : Computer Science Logic, 22nd International Workshop*, volume 5213 of *Lecture Notes in Comput. Sci.*, pages 416–430. Springer, 2008.

[32] Thomas Colcombet and Christof Löding. The non-deterministic Mostowski hierarchy and distance-parity automata. In Luca Aceto, Ivan Damgård, Leslie Ann Goldberg, Magnús M. Halldórsson, Anna Ingólfsdóttir, and Igor Walukiewicz, editors, *ICALP 2008 (2) : Automata, Languages and Programming, 35th International Colloquium*, volume 5126 of *Lecture Notes in Comput. Sci.*, pages 398–409. Springer, 2008.

[33] Thomas Colcombet and Christof Löding. Regular cost functions over finite trees. In *LICS 2010 : 5th Annual IEEE Symposium on Logic in Computer Science*, pages 70–79. IEEE Computer Society, 2010.

[34] Hubert Comon, Max Dauchet, Rémi Gilleron, Christoph Löding, Florent Jacquemard, Denis Lugiez, Sophie Tison, and Marc Tommasi. Tree automata techniques and applications, 2007.

[35] Françoise Dejean and Marcel-Paul Schützenberger. On a question of Eggan. *Information and Control*, 9(1) :23–25, 1966.

[36] John Doner. Tree acceptors and some of their applications. *J. Comput. Syst. Sci.*, 4(5) :406–451, 1970.

[37] Lawrence C. Eggan. Transition graphs and the star-height of regular events. *Michigan Math. J.*, 10 :385–397, 1963.

[38] Calvin C. Elgot. Decision problems of finite automata design and related arithmetics. *Trans. Amer. Math. Soc.*, 98 :21–51, 1961.

[39] Hugo Gimbert. Parity and exploration games on infinite graphs. In Jerzy Marcinkowski and Andrzej Tarlecki, editors, *CSL 2004 : Computer Science Logic, 18th International Workshop*, volume 3210 of *Lecture Notes in Comput. Sci.*, pages 56–70. Springer, 2004.

[40] Yuri Gurevich and Leo Harrington. Trees, automata, and games. In Harry R. Lewis, Barbara B. Simons, Walter A. Burkhard, and Lawrence H. Landweber, editors, *STOC 1982 : 14th Annual ACM Symposium on Theory of Computing*, pages 60–65. ACM, 1982.

[41] Kosaburo Hashiguchi. A decision procedure for the order of regular events. *Theoret. Comput. Sci.*, 8 :69–72, 1979.

[42] Kosaburo Hashiguchi. Limitedness theorem on finite automata with distance functions. *J. Comput. Syst. Sci.*, 24(2) :233–244, 1982.

[43] Kosaburo Hashiguchi. Regular languages of star height one. *Information and Control*, 53(3) :199–210, 1982.

[44] Kosaburo Hashiguchi. Representation theorems on regular languages. *J. Comput. Syst. Sci.*, 27(1) :101–115, 1983.

[45] Kosaburo Hashiguchi. Relative star height, star height and finite automata with distance functions. In Jean-Eric Pin, editor, *Formal Properties of Finite Automata and Applications*, volume 386, pages 74–88, 1988.

[46] Kosaburo Hashiguchi. Improved limitedness theorems on finite automata with distance functions. *Theoret. Comput. Sci.*, 72(1) :27–38, 1990.

[47] Thomas A. Henzinger and Nir Piterman. Solving games without determinization. In Zoltán Ésik, editor, *CSL 2006 : Computer Science Logic, 20th International Workshop*, volume 4207 of *Lecture Notes in Comput. Sci.*, pages 395–410. Springer, 2006.

[48] Szczepan Hummel and Michal Skrzypczak. The topological complexity of MSO+U and related automata models. *Fundam. Inform.*, 119(1) :87–111, 2012.

[49] Michael Kaminski and Nissim Francez. Finite-memory automata. *Theoret. Comput. Sci.*, 134(2) :329–363, 1994.

[50] J. A. Kamp. *Tense Logic and the Theory of Linear Order*. PhD thesis, Univ. of California, Los Angeles, 1968.

[51] Daniel Kirsten. Desert automata and the finite substitution problem. In Volker Diekert and Michel Habib, editors, *STACS 2004 : 21st International Symposium on Theoretical Aspects of Computer Science*, volume 2996 of *Lecture Notes in Comput. Sci.*, pages 305–316. Springer, 2004.

[52] Daniel Kirsten. Distance desert automata and the star height problem. *RAIRO - Theor. Inf. Appl.*, 3(39) :455–509, 2005.

[53] Daniel Kirsten. Some variants of the star height problem. In Filip Murlak and Piotr Sankowski, editors, *MFCS 2011 : Mathematical Foundations of Computer Science*, volume 6907 of *Lecture Notes in Comput. Sci.*, pages 19–33. Springer, 2011.

[54] Nils Klarlund. Progress measures, immediate determinacy and a subset construction for finite automata. *Annals of Pure and Applied Logic*, 69 :243–268, 1994.

[55] Stephen C. Kleene. Representation of events in nerve nets and finite automata. In Claude E. Shannon and John McCarthy, editors, *Automata Studies*, pages 3–42. Princeton University Press, Princeton, New Jersey, 1956.

[56] Daniel Krob. The equality problem for rational series with multiplicities in the tropical semiring is undecidable. *Internat. J. Algebra Comput.*, 4(3) :405–425, 1994.

[57] Manfred Kufleitner. The height of factorization forests. In Edward Ochmanski and Jerzy Tyszkiewicz, editors, *MFCS 2008 : Mathematical Foundations of Computer Science*, volume 5162 of *Lecture Notes in Comput. Sci.*, pages 443–454. Springer, 2008.

[58] Denis Kuperberg. Linear temporal logic for regular cost functions. In Thomas Schwentick and Christoph Dürr, editors, *STACS 2011 : 28th International Symposium on Theoretical Aspects of Computer Science*, volume 9 of *LIPIcs*, pages 627–636. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2011.

[59] Denis Kuperberg. *Étude de classes de fonctions de coût régulière*. PhD thesis, Université Paris Denis Diderot, 2012.

[60] Denis Kuperberg and Michael Vanden Boom. Quasi-weak cost automata : A new variant of weakness. In Supratik Chakraborty and Amit Kumar, editors, *FSTTCS 2011 : IARCS Annual Conference on Foundations of Software Technology and Theoretical Computer Science*, volume 13 of *LIPIcs*, pages 66–77. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2011.

[61] Denis Kuperberg and Michael Vanden Boom. On the expressive power of cost logics over infinite words. In Artur Czumaj, Kurt Mehlhorn, Andrew M. Pitts, and Roger Wattenhofer, editors, *ICALP 2012 (2) : Automata, Languages, and Programming - 39th International Colloquium*, volume 7392 of *Lecture Notes in Comput. Sci.*, pages 287–298. Springer, 2012.

[62] Orna Kupferman and Moshe Y. Vardi. The weakness of self-complementation. In Christoph Meinel and Sophie Tison, editors, *STACS 1999 : 16th International Symposium on Theoretical Aspects of Computer Science*, volume 1563 of *Lecture Notes in Comput. Sci.*, pages 455–466. Springer, 1999.

[63] Hing Leung. *An Algebraic Method for Solving Decision Problems in Finite Automata Theory*. PhD thesis, Pennsylvania State University, Department of Computer Science, 1987.

[64] Hing Leung. On the topological structure of a finitely generated semigroup of matrices. *Semigroup Forum*, 37 :273–287, 1988.

[65] Hing Leung. Limitedness theorem on finite automata with distance functions : An algebraic proof. *Theoret. Comput. Sci.*, 81(1) :137–145, 1991.

[66] Hing Leung and Viktor Podolskiy. The limitedness problem on distance automata : Hashiguchi's method revisited. *Theoret. Comput. Sci.*, 310(1-3) :147–158, 2004.

[67] Christof Löding. Finding deterministic subautomata in polynomial time. Personal communication, 2011.

[68] Donald A. Martin. Borel determinacy. *Ann. Math.*, 102 :363–371, 1975.

[69] Antoni Mazurkiewicz. Concurrent program schemes and their interpretations. Technical Report DAIMI Report PB 78, Aahrus University, 1977.

[70] Robert McNaughton. Testing and generating infinite sequences by a finite automaton. *Information and Control*, 9 :521–530, 1966.

[71] Robert McNaughton. The loop complexity of pure-group events. *Information and Control*, 11(1-2) :167–176, 1967.

[72] Robert McNaughton and Seymour Papert. *Counter-free Automata*. MIT Press, 1971.

[73] Satoru Miyano and Takeshi Hayashi. Alternating finite automata on ω-words. *Theoret. Comput. Sci.*, 32 :321–330, 1984.

[74] Andrzej W. Mostowski. Regular expressions for infinite trees and a standard form of automata. In *Symposium on Computation Theory*, pages 157–168, 1984.

[75] David E. Muller, Ahmed Saoudi, and Paul E. Schupp. Alternating automata. the weak monadic theory of the tree, and its complexity. In Laurent Kott, editor, *ICALP 1986 : Automata, Languages and Programming, 13th International Colloquium*, volume 226 of *Lecture Notes in Comput. Sci.*, pages 275–283. Springer, 1986.

[76] David E. Muller and Paul E. Schupp. The theory of ends, pushdown automata and second-order logic. *Theoret. Comput. Sci.*, 37 :51–76, 1985.

[77] David E. Muller and Paul E. Schupp. Alternating automata on infinite trees. *Theoret. Comput. Sci.*, 54 :267–276, 1987.

[78] David E. Muller and Paul E. Schupp. Simulating alternating tree automata by nondeterministic automata : New results and new proofs of the theorems of Rabin, McNaughton and Safra. *Theoret. Comput. Sci.*, 141(1&2) :69–107, 1995.

[79] John R. Myhill. Finite automata and representation of events. *Fund. Concepts in the Theory of Systems*, pages 57–624, 1957.

[80] Edward Nelson. Internal set theory : A new approach to nonstandard analysis. *Bulletin of the American Mathematical Society*, 83(6), 1977.

[81] Damian Niwiński. On fixed-point clones. In Laurent Kott, editor, *ICALP 1986 : Automata, Languages and Programming, 13th International Colloquium*, volume 226 of *Lecture Notes in Comput. Sci.*, pages 464–473. Springer, 1986.

[82] Damian Niwiński and Igor Walukiewicz. Relating hierarchies of word and tree automata. In Michel Morvan, Christoph Meinel, and Daniel Krob, editors, *STACS 1998 : 15th International Symposium on Theoretical Aspects of Computer Science*, volume 1373 of *Lecture Notes in Comput. Sci.*, pages 320–331. Springer, 1998.

[83] Damian Niwiński and Igor Walukiewicz. Deciding nondeterministic hierarchy of deterministic tree automata. *Electr. Notes Theor. Comput.*, 123 :195–208, 2005.

[84] Yves Péraire. Théorie relative des ensembles internes. *Osaka J. Math.*, 29 :267–297, 1992.

[85] Dominique Perrin. Les débuts de la théorie des automates. *Technique et science informatiques*, 14(4) :409–433, 1995.

[86] Dominique Perrin and Jean-Éric Pin. Semigroups and automata on infinite words. In J. Fountain, editor, *NATO Advanced Study Institute Semigroups, Formal Languages and Groups*, pages 49–72. Kluwer academic publishers, 1995.

[87] Michael O. Rabin. Decidability of second-order theories and automata on infinite trees. *Trans. Amer. Math. Soc.*, 141 :1–35, 1969.

[88] Michael O. Rabin. Weakly definable relations and special automata. In Yehoshua Bar-Hillel, editor, *Mathematical Logic and Foundations of Set Theory*, pages 1–23. North Holland, 1970.

[89] Michael O. Rabin and Dana Scott. Finite automata and their decision problems. *IBM J. Res. and Develop.*, 3 :114–125, April 1959.

[90] Abraham Robinson. *Non-standard analysis.* North Holland Publishing Co., 1966.

[91] Marcel-Paul Schützenberger. On an application of semigroups methods to some problems in coding. *Information Theory, IRE Transactions on*, 2(3) :47 –60, september 1956.

[92] Marcel-Paul Schützenberger. On the definition of a family of automata. *Information and Control*, 4 :245–270, 1961.

[93] Marcel-Paul Schützenberger. On finite monoids having only trivial subgroups. *Information and Control*, 8 :190–194, 1965.

[94] Olivier Serre. Parity games played on transition graphs of one-counter processes. In Luca Aceto and Anna Ingólfsdóttir, editors, *FoSSaCS 2006 : Foundations of Software Science and Computation Structures, 9th International Conference*, volume 3921 of *Lecture Notes in Comput. Sci.*, pages 337–351. Springer, 2006.

[95] Saharon Shelah. The monadic theory of order. *Ann. of Math. (2)*, 102(3) :379–419, 1975.

[96] Imre Simon. Limited subsets of a free monoid. In *FOCS 1978 : 19th Annual Symposium on Foundations of Computer Science*, pages 143–150. IEEE Computer Society, 1978.

[97] Imre Simon. Factorization forests of finite height. *Theoret. Comput. Sci.*, 72 :65–94, 1990.

[98] Imre Simon. On semigroups of matrices over the tropical semiring. *RAIRO - Theor. Inf. Appl.*, 28(3-4) :277–294, 1994.

[99] Terence Tao. Compactness and contradiction. Livre en préparation.

[100] James W. Thatcher and Jesse B. Wright. Generalized automata theory with an appliucation to a decision problem in second-order logic. *Math. Syst. Theory*, 2 :57–81, 1968.

[101] Wolfgang Thomas. Languages, automata and logic. In Gregorz Rozenberg and Arto Salomaa, editors, *Handbook of Formal Languages*, volume 3, chapter 7, pages 389–455. Springer, 1997.

[102] Szymon Toruńczyk. *Languages of profinite words and the limitedness problem.* PhD thesis, Warsaw University, 2011.

[103] Yaron Velner and Alexander Rabinovich. Church synthesis problem for noisy input. In Martin Hofmann, editor, *FoSSaCS 2011 : Foundations of Software Science and*

Computational Structures - 14th International Conference, volume 6604 of *Lecture Notes in Comput. Sci.*, pages 275–289. Springer, 2011.

[104] Klaus Wagner. Eine topologische charakterisierung einiger klassen regulären folgenmengen. *Elektron. Informationsverarb. Kybernet.*, 13 :473–487, 1977.

[105] Igor Walukiewicz. Pushdown processes : Games and model checking. In *CAV*, Lecture Notes in Comput. Sci., pages 62–74. Springer, 1996.

[106] Thomas Wilke. An Eilenberg theorem for ∞-languages. In *ICALP 91 : Automata, Languages and Programming, 18th International Colloquium*, volume 510 of *Lecture Notes in Comput. Sci.*, pages 588–599. Springer, 1991.

[107] Thomas Wilke. An algebraic theory for regular languages of finite and infinite words. *Int. J. Alg. Comput.*, 3 :447–489, 1993.

[108] Wiesław Zielonka. Infinite games on finitely coloured graphs with applications to automata on infinite trees. *Theoret. Comput. Sci.*, 200 :135–183, 1998.

Ce mémoire propose un panorama de la théorie des fonctions régulières de coût. Les fonctions régulières de coût forment une extension quantitative des langages réguliers. Les fonctions de coût sont des fonctions qui à chaque mot (ou arbre...) associent un entier ou l'infini. Elles sont considérées modulo une relation d'équivalence qui autorise une certaine distorsion des valeurs exactes, mais garantit la préservation de l'existence de bornes.

Les fonctions régulières de coût forment une sous-classe des fonctions de coût qui admet de nombreuses caractérisations effectivement équivalentes, en termes de logiques, d'automates, d'algèbre ou d'expressions régulières. Des résultats de décidabilité en sont déduits.

Ces résultats étendent d'une part les résultats classiques sur les langages réguliers, et d'autre part les travaux de Hashiguchi, Simon, Leung et Kirsten sur les automates de distance, le semi-anneau tropical et le problème de la hauteur d'étoile.

Ce document décrit l'état actuel des connaissances sur ces objets dans le cas des mots finis et infinis et des arbres finis et infinis.

This thesis gives a broad picture of the theory of regular cost functions. Regular cost functions are quantitative extensions to regular languages. A cost function is a function from words (or trees...) to the non-negative integers or infinity. These functions are considered modulo an equivalence relation that allows some distortion of the exact values, but preserves the existence of bounds.

Regular cost functions form a sub-class that admits many effectively equivalent characterisations in terms of logic, automata, algebra and regular expressions. Some decidability results are deduced.

These results extend on the one hand the classical results on regular languages, and on the other hand the works of Hashiguchi, Simon, Leung, and Kirsten concerning distance automata, the tropical semiring and the star-height problem.

This document describes the current knowledge about regular cost functions for finite words, infinite words, finite trees and infinite trees.